Faculty of Technology
School of Computer Science and Informatics

Advanced Threat Intelligence: Interpretation of Anomalous Behavior in Ubiquitous Kernel Processes

PhD Thesis

Robert Luh

Submitted in fulfillment of the requirements
for the degree of Doctor of Philosophy

July, 2019

Abstract

Targeted attacks on digital infrastructures are a rising threat against the confidentiality, integrity, and availability of both IT systems and sensitive data. With the emergence of advanced persistent threats (APTs), identifying and understanding such attacks has become an increasingly difficult task. Current signature-based systems are heavily reliant on fixed patterns that struggle with unknown or evasive applications, while behavior-based solutions usually leave most of the interpretative work to a human analyst.

This thesis presents a multi-stage system able to detect and classify anomalous behavior within a user session by observing and analyzing ubiquitous kernel processes. Application candidates suitable for monitoring are initially selected through an adapted sentiment mining process using a score based on the log likelihood ratio (LLR). For transparent anomaly detection within a corpus of associated events, the author utilizes star structures, a bipartite representation designed to approximate the edit distance between graphs. Templates describing nominal behavior are generated automatically and are used for the computation of both an anomaly score and a report containing all deviating events. The extracted anomalies are classified using the Random Forest (RF) and Support Vector Machine (SVM) algorithms. Ultimately, the newly labeled patterns are mapped to a dedicated APT attacker–defender model that considers objectives, actions, actors, as well as assets, thereby bridging the gap between attack indicators and detailed threat semantics. This enables both risk assessment and decision support for mitigating targeted attacks.

Results show that the prototype system is capable of identifying 99.8% of all star structure anomalies as benign or malicious. In multi-class scenarios that seek to associate each anomaly with a distinct attack pattern belonging to a particular APT stage we achieve a solid accuracy of 95.7%. Furthermore, we demonstrate that 88.3% of observed attacks could be identified by analyzing and classifying a single ubiquitous Windows process for a mere 10 seconds, thereby eliminating the necessity to monitor each and every (unknown) application running on a system.

With its semantic take on threat detection and classification, the proposed system offers a formal as well as technical solution to an information security challenge of great significance.

Declaration

This is to certify that:

(i) the thesis comprises only my original work towards the PhD except where indicated,

(ii) due acknowledgement has been made in the text to all other material used,

(iii) the thesis joins the original work published in at least three peer-reviewed journals with a critical and theoretical narrative of approximately 20,000 words (PhD by concurrent publication).

Robert Luh

Publications

During the course of this project, a number of peer-reviewed publications and public presentations have been made based on the work presented in this thesis. They are listed here for reference.

- [187] **Robert Luh, Stefan Marschalek, Manfred Kaiser, Helge Janicke, and Sebastian Schrittwieser**. Semantics-aware detection of targeted attacks: a survey. Journal of Computer Virology and Hacking Techniques, pages 1–39. Springer, 2016.
- [188] **Robert Luh, Sebastian Schrittwieser, and Stefan Marschalek**. TAON: An ontology-based approach to mitigating targeted attacks. In Proceedings of the 18th International Conference on Information Integration and Web-based Applications & Services (IIWAS). ACM, 2016.
- [189] **Robert Luh, Sebastian Schrittwieser, Stefan Marschalek, Helge Janicke, and Edgar Weippl**. Design of an anomaly-based threat detection & explication system. In Proceedings of the 3rd International Conference on Information Systems Security and Privacy (ICISSP), pages 397–402. SCITEPRESS, 2017.
- [190] **Robert Luh, Sebastian Schrittwieser, and Stefan Marschalek**. LLR-based sentiment analysis for kernel event sequences. In Proceedings of the 31st International Conference on Advanced Information Networking and Applications (AINA), pages 764–771. IEEE, 2017.
- [191] **Robert Luh, Gregor Schramm, Markus Wagner, and Sebastian Schrittwieser**. Sequitur-based inference and analysis framework for malicious system behavior. In Proceedings of the 3rd International Conference on Information Systems Security and Privacy (ICISSP), pages 632–643. SCITEPRESS, 2017.
- [192] **Robert Luh, Sebastian Schrittwieser, Stefan Marschalek, Helge Janicke, and Edgar Weippl**. Poster: Design of an anomaly-based threat detection & explication system. In Proceedings of the 22nd ACM on Symposium on Access Control Models and Technologies (SACMAT), pages 119–120. ACM, 2017.
- [193] **Robert Luh, Gregor Schramm, Markus Wagner, Helge Janicke, and Sebastian Schrittwieser**. SEQUIN: a grammar inference framework for analyzing malicious system behavior. Journal of Computer Virology and Hacking Techniques, pages 1–21. Springer, 2018.
- [194] **Robert Luh, Marlies Temper, Simon Tjoa, and Sebastian Schrittwieser**. APT RPG: Design of a gamified attacker/defender meta model. In Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP). SCITEPRESS, 2018.

- [195] **Robert Luh, Marlies Temper, Simon Tjoa, Sebastian Schrittwieser, and Helge Janicke**. PenQuest: A Gamified Attacker/Defender Meta Model for Cyber Security Assessment and Education. Journal of Computer Virology and Hacking Techniques. Springer, *submitted*, 2018.
- [196] **Robert Luh, Helge Janicke, and Sebastian Schrittwieser**. AIDIS: Detecting and Classifying Anomalous Behavior in Ubiquitous Kernel Processes. Journal of Computers and Security (COSE). Elsevier, 2019.
- [201] **Stefan Marschalek, Robert Luh, Manfred Kaiser, and Sebastian Schrittwieser**. Classifying malicious system behavior using event propagation trees. In 17th International Conference on Information Integration and Web-based Applications & Services (IIWAS). ACM, 2015.
- [202] **Stefan Marschalek, Robert Luh, and Sebastian Schrittwieser**. Endpoint data classification using Markov chains. In 2017 International Conference on Software Security and Assurance (ICSSA), pages 56–59. IEEE, 2017.

Preface

This thesis is a cumulative work combining 4 peer-reviewed journal papers as well as one conference paper for concurrent publication. The incorporated works are at the core of Chapters 2 as well as Chapters 4 to 7 and have only been slightly adapted in content and formatting to fit the dissertation format. The original papers are:

- **Chapter 2**: Robert Luh, Stefan Marschalek, Manfred Kaiser, Helge Janicke, and Sebastian Schrittwieser. Semantics-aware detection of targeted attacks: a survey. Journal of Computer Virology and Hacking Techniques, pages 1–39. Springer, 2016.
- **Chapter 4**: Robert Luh, Sebastian Schrittwieser, and Stefan Marschalek. LLR-based sentiment analysis for kernel event sequences. In Proceedings of the 31st International Conference on Advanced Information Networking and Applications (AINA), pages 764–771. IEEE, 2017.
- **Chapter 5**: Robert Luh, Gregor Schramm, Markus Wagner, Helge Janicke, and Sebastian Schrittwieser. SEQUIN: a grammar inference framework for analyzing malicious system behavior. Journal of Computer Virology and Hacking Techniques, pages 1–21. Springer, 2018.
- **Chapter 6**: Robert Luh, Marlies Temper, Simon Tjoa, Sebastian Schrittwieser, and Helge Janicke. PenQuest: A Gamified Attacker/Defender Meta Model for Cyber Security Assessment and Education. Journal of Computer Virology and Hacking Techniques. Springer, *submitted*, 2018.
- **Chapter 7**: Robert Luh, Helge Janicke, and Sebastian Schrittwieser. AIDIS: Detecting and Classifying Anomalous Behavior in Ubiquitous Kernel Processes. Journal of Computers and Security (COSE). Elsevier, 2019.

The central premises, statements, and conclusions of above papers remain unaltered, as mandated by DMU regulations. In accordance to the same rules, the thesis author is responsible for producing the draft as well as at least 75% of the final content of each incorporated paper. The respective confirmation of primary authorship has been signed by all co-authors and can be found in the appendix.

Please note that the author refers to himself in the plural throughout the document. This is done to be in line with the formulation used in the individual papers and does not imply the involvement of other contributors. With the exception of some minor parts of the aforementioned papers, no third party was involved in the writing of this document.

Acknowledgements

First of all I would like to thank my doctoral supervisor Dr. Helge Janicke for his guidance and the opportunity to work on this topic of personal interest. Without Helge's advice, many of the challenges – in both research and bureaucracy – might have appeared insurmountable.

Secondly, I would like to express my sincere gratitude to my colleague Thomas Brandstetter, who served as both ambassador and liaison to De Montfort University. It was his input that made me consider pursuing a PhD in the first place.

Most importantly, none of this would have been possible without the love and patience of my wife Martina and my parents Mila and Heiner, to whom I would like to express my heart-felt gratitude. Thank you for bearing with me and for listening to my endless rambling about models, malware, and machine learning. A big "thanks, mate" also to my brother Martin who regularly reminded me that life is not just about computers.

I would also like to thank my fellow colleagues Dr. Paul Tavolato and Dr. Sebastian Schrittwieser who were instrumental in planning and kick-starting the associated research projects. My gratitude extends to Dr. Maria Gabriela Ondrejkovics Fernandes, Gernot Kohl and Johann Haag, who greatly supported me in their role as university management here at St. Pölten University of Applied Sciences.

Finally, the financial support by the Christian Doppler Research Association, the Austrian Federal Ministry for Digital and Economic Affairs, and the National Foundation for Research, Technology and Development is gratefully acknowledged.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Contents

1.1 Motivation

IT systems are threatened by an ever-growing number of cyber-attacks. With the emergence of Advanced Persistent Threats (APTs), the focus shifted from off-the-shelf malware to multipartite attacks that are tailored to specific organizations or systems. These targeted threats are driven by varying motivations, such as espionage or sabotage, and often cause significantly more damage [305].

Several cases have shown that targeted attacks can remain undiscovered by their victims for many months or even years [71, 118, 126, 159, 307]. The prime example, Stuxnet, which targeted programmable logic controllers (PLCs) of sensitive industrial systems, was active for at least 3 years until discovery [294]. According to a Symantec study [95], Stuxnet infected close to 100,000 systems across 115 countries. Its quasi successor, Duqu, also targeted industrial control systems (ICS), gathering sensitive information in at least eight countries [51, 152]. On the espionage side, the Regin Trojan is believed to have been used for global, systematic campaigns since at least 2008 [304]. Other examples include Flame [153], Mahdi [279], and Gauss [154]. These strains are currently used for cyber-espionage in Middle Eastern countries and, depending on the variant, are capable of stealing passwords and cookies, recording network traffic, keystrokes, microphone audio, and even entire Skype conversations [214].

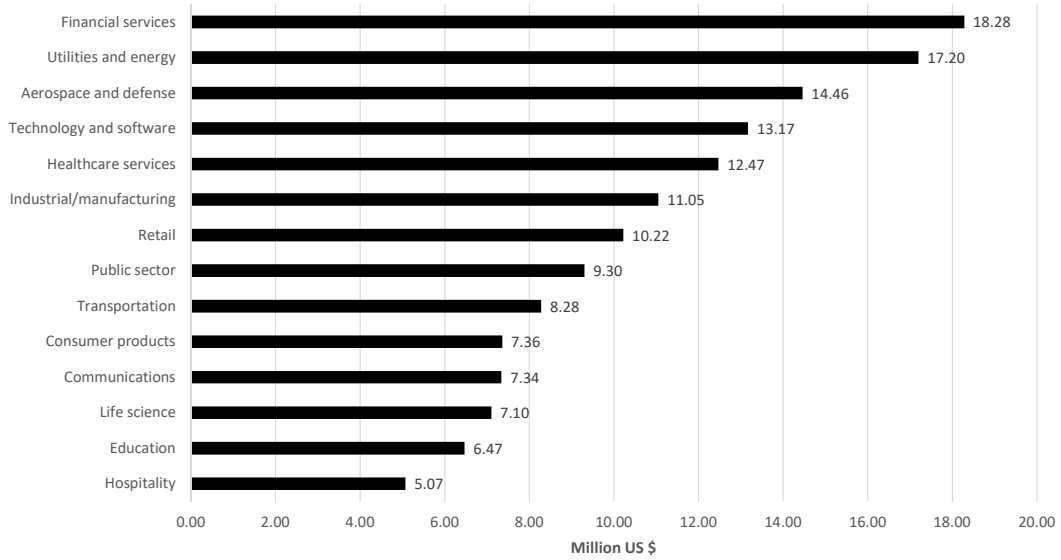


Figure 1.1: Average annualized cost of cyber-attacks by industry sector [136]. The study investigated a total of $n = 254$ companies.

APTs are increasingly affecting less prominent targets as well. In 2013 alone, *"economic espionage and theft of trade secrets cost the American economy more than $19 billion. Over the past 4 fiscal years, the number of arrests related to economic espionage and theft of trade secrets overseen by the FBI's Economic Espionage Unit has almost doubled, indictments have more than tripled, and convictions have increased sixfold. Halfway through fiscal year 2013, the number of open investigations is running more than 30 percent above the total from 4 years ago"* [231]. In the 2017 Official Annual Cybercrime Report by Cybersecurity Ventures [229], analysts speak of an estimated $6 trillion of annual damage that will be caused by cyber-attacks by 2021. A 2017 study by the Ponemon Institute [136] identifies the financial sector as the most affected by cybercrime, closely followed by the utilities and energy sectors. Figure 1.1 breaks down current costs by industry.

While APTs use malware like most conventional attacks, the level of complexity and sophistication of the malicious programs is usually higher. This is problematic since defensive measures offered by security vendors often utilize primarily *signature-based* systems, which are effective in the defense against known exploit carriers or ill-considered user actions but struggle with hitherto unknown malware [80]: Traditional misuse detection relies heavily on signature databases that have to be updated whenever a new attack technique or sample is discovered. For emerging threats in particular, such binary patterns of the involved malware are unlikely to exist at the time of attack. Poly- and metamorphic techniques [237] additionally obfuscate malicious software by creating self-altering malware variants that sport differing static appearance for cryptor and payload. Mimicry attacks, on the other hand, might substitute system calls, interweave instructions, or attempt to avoid generating observable events at all, which increases the complexity of dynamic detection efforts [329]. Lastly, the multi-stage nature of

APTs makes it generally difficult to interpret findings without semantic context, as certain systemic events can only be considered malicious in conjunction with other activity [187].

While APTs are undoubtedly among the greatest digital threats to organizations, off-the-shelf malware and "hacks of opportunity" remain an issue for individuals and companies alike. The frequency of such semi-targeted attacks against computers with Internet connectivity is still increasing: a Clark School study at the University of Maryland [68] recorded a mean of 2,244 attacks on each participating machine – which amounts to one attack every 39 seconds. While such threats are not typically part of sabotage or espionage efforts, computers compromised this way may still be abused as proxy attackers in APT campaigns.

Targeted or not, modern cyber-threats are no longer limited to a single malware executable (i.e. sample) but often comprise multi-stage attacks that are difficult to spot using only file- and signature-based malware detection systems. Therefore, it is necessary to explore novel techniques for threat intelligence and APT detection that are augmented with contextual information and provide resilience to various stealth techniques. The use of behavioral anomaly detection and classification allows for a "defense in depth" approach to attack mitigation: threats identified as anomaly can be blocked immediately, followed by intelligent attack classification and the generation of new alert patterns describing malicious actions. Behavior-based solutions are a promising means to identify and learn from such behavior. No matter the stealth techniques employed, the attacker will eventually execute his or her action on target – be it data theft, hijacking or sabotage. Anomalies signifying a deviation from a known behavioral baseline can then be used to detect the threat in its early stages. However, most existing systems do not provide the offending behavioral data to the analyst and contribute little to its interpretation. We argue that closing the resulting semantic gap is a vital next step in holistic (IT) system threat mitigation.

The following details additional information about the problems considered, highlights key contributions and introduces related background information. All research questions and hypotheses are discussed in detail, followed by information on the structure of the thesis as a whole.

1.2 Problem Statement

We have identified three key problems of current threat detection methods and systems that need to be addressed for improved APT detection. These encompass a) the limitations of pattern-based systems, b) the current focus on known binary samples for behavioral analysis and the general lack of complementary observation points, as well as c) shortcomings of current systems in terms of classification and interpretation of detected threats.

1.2.1 Problem 1: Limitations of Pattern-Based Detection

Pattern-based threat detection relies on existing knowledge about utilized software and attack techniques that is unlikely to exist for novel (targeted) attacks. Furthermore, patterns are more susceptible to obfuscation and mimicry attacks.

Threat detection is at the heart of modern cyber-defense. In this thesis, we review memory-based approaches, numerous behavioral detection and analysis systems, function call monitoring solutions, host-based intrusion detection systems, and more. Like many IDS and anti-virus (AV) products, most of these solutions have in common their reliance on binary signatures or other predefined patterns to identify malicious characteristics [5].

Pattern (misuse) detection is based on predefined templates. Knowledge of an attacker's methods or the expected consequences of an attack are encoded into behavioral patterns or strings that can be found by an IDS or AV solution looking for their occurrence [170]. Examples include sequences of suspicious opcalls, the use of certain API functions, network packet payloads, or the existence of specific characters or binary patterns within a file. This approach is traditionally referred to as signature-based detection, which is known to have several shortcomings [54]:

Firstly, obfuscation techniques commonly utilize polymorphic or metamorphic mutation to generate an ever-growing number of malware variants that are functionally identical but different in appearance. Similarly, function or opcall patterns may be circumvented through substitute commands or the targeted induction of errors.

Secondly, many common signature-based tools only detect malware which has already been identified and analyzed by e.g. an AV vendor. New species or hitherto unknown variants are overlooked if no corresponding pattern exists in the database at the time of the attack.

1.2.2 Problem 2: Lack of Complementary Observation Points

Single points of observation are insufficient, since modern attacks comprise both local and network activity. Furthermore, sample-based analysis often builds on existing suspicions about a certain binary, which will not exist in the case of an unknown attack. Evasion techniques are more likely to skew the results of on-demand sample analysis systems.

APTs and advanced malware manifest as multipartite attacks that can no longer be identified or mitigated at one point of observation alone. Security information and event management (SIEM) systems [164] go in the right direction by offering event correlation across data sources, but are often nontransparent in their functionality. Furthermore, we argue that system and application logs are not sufficient to truly understand kernel-level behavior. On the analysis side, malware sandboxes offer on-demand investigation for

select suspicious binaries but typically do not encompass host-based IDS functionality for continuous system monitoring, which would be required to detect activity stemming from unknown sources.

The majority of analysis systems focus on suspected malware samples that are often analyzed independently from the rest of the system [330]. Suspicious files or processes may prompt the starting of an analysis sandbox that keeps the sample running for a predetermined amount of time – be that manually by an operator or automatically when passing on an attachment from an IDS to an analysis environment. Such sandboxes are the foundation for many a threat detection implementation. Depending on a system's capabilities and requirements [330], these environments may be physical machines, virtual machines [115], or emulated systems [29].

In both cases, determining the nature of the threat requires advance knowledge about which process to observe and creates problems with sandbox detection and evasion techniques [139]. For example, if an attack solely utilizes system processes without dropping additional binaries, sample-focused systems may not trigger at all. Even if they do, various evasion techniques can cause problems: Frequently, malware samples are coded to delay their execution until most sandboxes time out, or will utilize detection mechanisms that seek to identify the analysis environment. What follows is a change in behavior that will likely hide relevant information from the analyst or crash the sample altogether.

Furthermore, most attacks and malware infections include activity on both the affected host and the connected network [28, 120, 232, 356]. Payload downloads, command and control (C2) traffic, and general exfiltration activity will result in kernel events representative for file system as well as network activity. Focusing on only one or the other is not sufficient to understand the adversary's objective.

1.2.3 Problem 3: Insufficient Interpretation of Malicious Events

> **The interpretation of malicious activity is currently lacking in regards to the identification of adversary techniques and objectives, making it difficult to understand advanced attacks.**

Event interpretation and event-to-goal mapping is currently lacking in depth, accuracy, and automation, which makes understanding advanced attacks difficult. Most classification systems are either not transparent in how they operate or do not offer much insight into the inner workings of a malicious procedure or program. Pattern-based detection tools usually feature a manually populated, significantly limited database of events that maps isolated behavior to e.g. a descriptive keyword. This is often not enough to truly understand the attack as a whole. Anomalies, on the other hand, usually offer little other than a threshold that, when exceeded, will trigger an alert.

The semantics (meaning) of an attack is an important factor for threat identification and analysis [187]. In the age of APTs, it is prudent to move away from black box

systems that present their findings as vague numbers and to work towards explainability. Unfortunately, most existing systems do not disseminate the offending behavioral data to the analyst and contribute little to an attack's interpretation. If they do, semantic enrichment is often done by means of fixed patterns similar to misuse detection scenarios, which is faced with the same range of problems. Of the 60 solutions surveyed as part of our literature review, only 13 potentially contribute to the detection of complex targeted attacks. The remainder has been developed with common threats in mind or present purely formal systems that are not tied to actual observables. We argue that closing the resulting semantic gap between concrete patterns/anomalies and semantic attack properties such as motivation, goal, involved actors, targeted system, and specific techniques employed is a vital next step in holistic (IT) system threat mitigation.

Current models for semantic enrichment are similarly limited. In the literature, the complex nature of targeted attacks and the mapping of specific exploits to a broader goal is often depicted as a linear multi-stage process. For example, Giura and Wang [113] and the cyber kill chain by Hutchins et al. [133] expand on the military concept of target engagement and define typical phases of an attack. While such models can often be used as a starting point for threat interpretation, they do not consider the intricacies of an attack in its entirety.

1.3 Research Questions and Hypotheses

We argue that behavioral analysis of dynamically captured data is key to understanding attack semantics, since static or pattern-based analysis requires knowledge of a file in questions and observes only predefined properties. Based on this premise, we formulate the following key research questions:

1. **How can advanced targeted attacks be comprehensively modeled in preparation for semantic enrichment?** We argue that a complete model of actor behavior ready for semantic annotation is vital for any attack interpretation effort. By answering this question we provide the foundation for closing the semantic gap between data and threat description.
2. **How can suspicious system behavior be accurately identified without relying on predefined patterns?** Since we have identified pattern-based systems as limited in some regards, it becomes necessary to research anomaly-based alternatives. The inherent identification of application behavior and its accurate classification into benign and malicious activity is at the core of our research.
3. **How can system anomalies be mapped to specific attacks?** Here, we tie sub-questions 1 and 2 together by researching mechanisms for the semantic enrichment of monitored (IDS) data such as behavioral anomalies.

Answering the three above research questions provides us with the tools for creating an intrusion detection and interpretation system unburdened by the problems of pattern-

based and sample-focused solutions that are currently in widespread use. The listed questions go hand in hand with the central hypotheses of the dissertation:

- **The detection of APTs can be improved by using semantics-aware techniques and tools.** We argue that current detection solutions are largely insufficient to recognize the wide range of targeted attacks currently conducted in the wild. It will become easier to understand and mitigate modern threats by employing semantics-aware techniques and methods.
- **System anomalies describing attacker behavior are more feasible to use in a holistic system/network environment than fixed misuse scenarios.** Following the problem statement, we hypothesize that anomaly detection approaches are better suited to detecting host and network attacks than maintenance-heavy patterns stored in a signature database.
- **Knowledge extraction followed by model-based attack explication is a viable approach to understanding targeted threats.** The interpretation of previously determined anomaly data (i.e. events that constitute a deviation) by mapping it to a dedicated attacker/defender model can, in conjunction with automated data classification, be used to explain threats in a comprehensible manner.
- **Observing ubiquitous OS kernel processes is a feasible alternative to sample-focused analysis.** We argue that assessing omnipresent processes in an operating system is an improvement over solely sample-focused analysis. Accuracy permitting, this approach could drastically reduce the number of local observation points while making an IDS more resilient to evasion and obfuscation.
- **Considering traces of abstracted behavior is more effective than observing raw API calls.** Using abstracted kernel activity instead of raw API calls reduces the overall vocabulary of observed events and promises increased accuracy.
- **Classifying anomalies is preferable to classifying entire traces of unknown behavior.** Classification algorithms used to categorize types of attacks promise improved accuracy and comparable performance when used on pre-identified anomalies instead of full system traces depicting all exhibited (benign and malicious) activity.

Our hypotheses are supported by our extensive literature review, as well as previous research into malware behavior and intrusion detection. All research questions are centrally answered in Chapter 8.

1.4 Structure of the Work

The remainder of this document is structured as follows: Chapter 2 provides background information and details the literature review process for the thesis. It identifies,

evaluates, and scores 60 papers and articles that introduce methods, models, frameworks, formalisms, or systems which could potentially contribute to the defense against APTs and other multi-stage cyber-attacks. Chapter 2 also presents our initial work into an APT defense framework and introduces a design checklist that was ultimately used to develop the AIDIS system. In Chapter 3, we discuss methodology, research and experiment design, data selection, and procedures used in this work.

Chapters 4 through 7 discuss the core components of the anomaly interpretation system. Initially, sentiment extraction from structured kernel event data is detailed, presenting an LLR-based approach to anomaly detection and identification for the most meaningful ubiquitous processes in the Windows operating system. In Chapter 5, formal definition of adversarial behavior is discussed. Here, the 'SEQUIN' grammar inference system for unsupervised anomaly detection, knowledge discovery, visualization, and event compression is explained. Chapter 6 takes a step back and provides a complete meta model for describing cyber-attacks in great detail. This foundation for our subsequent anomaly explication approach is grounded on a non-cooperative, imperfect incomplete information strategy game with a wide range of additional applications. Finally, Chapter 7 describes the AIDIS system as a whole and evaluates its functionality with a large set of real-world behavioral process data.

The system's overall evaluation is summarized in Chapter 8, where we also discuss implications, improvement potential, practical implementation, and suggested future work.

1.5 Summary

This thesis proposes anomaly-based threat detection revolving around the abstracted behavior of ubiquitous kernel processes. Anomaly detection is based on the premise that malicious activity manifests as abnormality that can be identified through measuring the variance of certain key metrics such as behavior deviating from a defined baseline (i.e. known nominal behavior). In our system, threat identification is performed using dynamic, behavior-based analysis, which is expected to be far more resilient to modern obfuscation and anti-analysis techniques than signature-based alternatives [330].

Our research seeks to further endpoint attack detection by reducing the reliance on patterns in general: All components of AIDIS forgo fixed, predefined sequences to define malicious behavior and instead use several novel approaches to flexible and transparent anomaly detection for captured operating system events. In response to the known performance issues of behavioral systems, we implemented grammar-based event compression to reduce the processing times of more expensive computations.

Our proposed system uses two observation points (local host and network) that are automatically correlated to create one single data basis for subsequent formalization and anomaly detection. No advance knowledge is required about any malware binaries dropped on the system. This is achieved by identifying and monitoring ubiquitous kernel

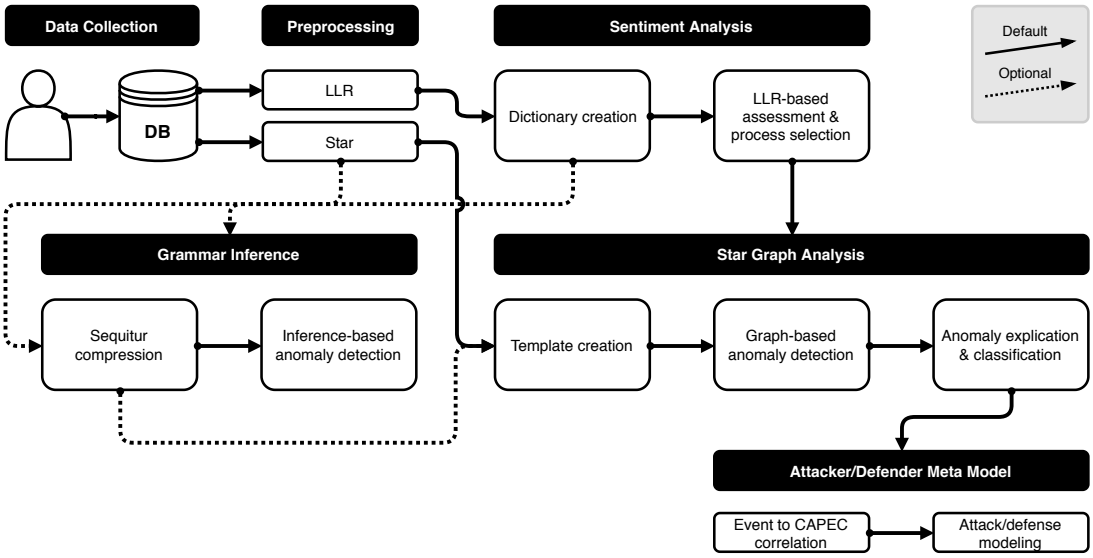


Figure 1.2: High-level overview of the AIDIS system.

processes, which are always present in a particular version of the operating system. We argue that execution delay mechanisms are not effective against kernel process observation, since any deviating activity of the OS process would constitute an anomaly no matter the initial execution time of the core malware.

Each component developed as part of AIDIS is transparent in its operation: No black-box anomaly detection is in use, always allowing analysts to fully understand machine learning decisions by presenting concrete patterns or anomaly descriptions contributing to a verdict or score. Anomaly classification is implemented through a set of generic competency questions and evaluated through established machine learning algorithms. The determined attack classes are mapped to a novel APT attacker–defender model that considers actions, actors, as well as assets and mitigating controls, thereby enabling decision support and contextual interpretation of past and ongoing attacks. Figure 1.2 depicts the proposed multi-stage system in its entirety.

In summary, this work contributes by:

- Presenting a sentiment-like analysis system for initial string-based anomaly detection and malicious process identification (Chapter 4);
- Providing flexible unsupervised anomaly detection for dedicated intrusion detection scenarios, as well as an effective compression system for arbitrary system event data (Chapter 5);
- Constructing a gamified attacker–defender meta model serving as recommendation system for cyber-attack mitigation (Chapter 6);
- Presenting and evaluating the prototype of an ‘Advanced Intrusion Detection and Interpretation System’ able to accurately classify anomalies by their APT stage and attack pattern (Chapter 7).

Chapter 2

Literature Review

Contents

2.1 Introduction

The sophisticated and multipartite nature of modern cyber-attacks is one of the main reasons why a defender needs to consider host-based, network-based and hybrid monitoring tools when it comes to capturing suspicious events that can later be analyzed and interpreted. While there are numerous solutions that focus on specific attack aspects such as malware infection or network intrusion, only few consider the greater picture. In order to truly understand a targeted attack, it becomes necessary to not only focus on malicious software or intrusion detection but to use every suitable tool at one's disposal. This chapter evaluates existing research retrieved from six academic search engines using an incremental search and refinement process utilized with a view to integrating the results into a holistic framework for APT detection and assessment. Existing models and technologies were methodically categorized by their capabilities, purpose, as well as several operational parameters with the aim to establish their applicability to a general concept of an APT defense framework presented that is the foundation of the system introduced in this thesis.

The remainder of the chapter is structured as follows: In Section 2.2 we introduce the concept of APTs, define frequent terms, and provide some background on models and use cases. Section 2.3 specifies the research questions, review method, and various literature collection criteria. In Section 2.4 we introduce the categorization schema used in the review process. Section 2.5 provides specific background and reviews the selected papers. Section 2.6 contains a comprehensive comparison charts and breaks down the results. In Section 2.7, we discuss our findings and present the concept of a defense framework, which became the design foundation of AIDIS.

2.2 APT Taxonomy and Semantics

2.2.1 Modeling Advanced Persistent Threats

APTs are not necessarily limited to a tailored piece of malware, but often encompass various stages with their own respective methods [133, 179]. For example, spearfishing and malicious websites are common approaches to delivering a malicious payload to a target system [294, 185]. RATs embedded in a Trojan horse are then often used to take control of a machine. In addition, targeted attacks reportedly employ zero-day vulnerabilities [34, 294].

In order to better understand the complex nature of targeted attacks and to map specific exploits to a broader goal, APTs are often modeled as a multi-stage process. Hutchins et al. [133] expand on the military concept of target engagement and define seven typical phases, as seen in Figure 2.1: (1) *Reconnaissance*, such as network scans, mapping procedures, employee profiling, and search for suitable zero-day exploits, (2) *weaponization*, i.e. the development of targeted malware and the set-up of malicious services, (3) *delivery* via various channels, (4) *exploitation*, such as the activation of

Figure 2.1: APT phases as defined by [133]

malware and the use of the previously weaponized payload to exploit a vulnerability, (5) *installation*, i.e. establishing a persistent presence, (6) *command & control* communication, and (7) *actions on objective*, which include the primary malicious task as well as exfiltration and clean-up activities.

Giuara and Wang [113] as well as De Vries et al. [71] use similar kill chain models to describe APTs – only the naming and some stage boundaries differs. Independent of taxonomy, APTs are complex attacks that, unlike bulk malware or fire-and-forget network attacks, carefully consider the target's system environment, security measures, and assets.

Adversary actions in each of the phases are possibly countered by a number of defense mechanisms. For example, the installation on the target system might be detected by a dedicated host-based intrusion detection system (HIDS). Each of the models helps to plan network defense measures as well as illustrates some data providers that are

promising to use in an attack detection system. This is the reason why we decided to include solutions from different domains instead of just malware detection tools.

2.2.2 Semantics-aware and Semantics-based Approaches

The semantics of an attack is an important factor for its identification and analysis. Generally speaking, semantics is the study of meaning. It is widely used in both technical and non-technical fields such as linguistics, philosophy, and information theory. When applied to computer science, it is often synonymous with determining execution paths of a program. In that case, the meaning of an artificial programming language as opposed to a natural language is evaluated. The mathematical model of computation is defined using one of the following techniques [344, 62]:

Denotational semantics describes the formalization of programming languages into mathematical objects. This includes the translation of a phrase into another, strictly formal language. *Operational semantics* encompasses the description of the execution and its correctness by e.g. describing machine state transitions. *Axiomatic semantics* describes meaning and proves correctness through rules of inference that map input to output properties. It is important to note that these strands are highly dependent on each other and are often used in concert.

In this document we differentiate semantics-aware and semantics-based approaches. We define *semantics-aware* solutions as considerate of the goal, means, and specifics of an attack. The overall premise can be summarized as the analysis of meaning of an attack. Based on this central aspect, we compiled a list of prerequisites that need to be met in order for a defense measure to be considered semantics-aware:

- The solution needs to counteract specific objectives or the overall goal of the attacker by e.g. detecting or preventing an attack or by helping to determine what the adversary wants to achieve (see primary category G in Section 2.4 for more information);
- The solution has to investigate, analyze, detect, or extract the technical means (i.e. techniques) of an attack or attack action through e.g. anomalies or patterns in behavior or code (see primary categories D and A in Section 2.4);
- The techniques, goals, or methods investigated can be mapped to one of the seven APT phases depicted in Figure 2.1 (primary category APT).

Semantics-based approaches usually revolve around four key topics we identified as representative for the semantic domain in general: activity context, ontology design, data abstraction, and correlation. As works in these fields are often more universal in nature, the above restrictions do not strictly apply. Instead, we opted to include security solutions that facilitate the development of models, languages and formal definitions or that determine the correctness of information through rules of e.g. inference. We argue that they, while perhaps not directly applicable to a particular scenario, are sufficiently adaptable to benefit the defense against advanced cyber-attacks.

Unlike semantics-aware approaches, the semantics-based category closely adheres to the type definitions introduced at the beginning of this section. For the literature review in Section 2.5, we categorized each paper in accordance to above definitions: Semantics-aware, semantics-based: denotational, operational, or axiomatic.

2.3 Review Method

This study is based on the guidelines for systematic literature reviews by Barbara Kitchenham [158], with some deviations from her guidelines. These include the omission of papers beyond the 50th result of each respective search engine, the post-review exclusion of articles with a low total score, and less reliance on search process automation.

The main goal of this chapter is to systematically review available literature on the topic of host-based, network-based, and multi-source detection of cyber-attacks with a focus on semantics-aware and semantics-based approaches. The surveyed papers are evaluated by their applicability to the domain of targeted attacks.

2.3.1 Research Questions

Specifically, there are a number of research questions addressed by this review:

1. Which models, frameworks, formal definitions, and tools exist to describe information system attacks?
2. Which semantics-aware and semantics-based tools and techniques exist to detect and evaluate such attacks?
3. What are promising approaches to APT detection and how can they be classified?
4. What kind of information is required to identify targeted attacks and how could it help to get a more complete picture of an incident?

To address R1 and R2, we have conducted a search of several scientific databases. This process is detailed in the next subsection. To satisfy R3, we came up with a list of criteria and categories to classify each identified solution. These criteria were combined into structured overview tables that are thoroughly explored in Section 2.6. In response to R4, Section 2.7 conceptually combines the benefits of solutions from several review categories and sketches an ideal defense framework.

2.3.2 Search Process

The search process included a manual keyword search on IEEE, ACM, Scopus, Web of Science, Academic Search Premier and Google Scholar. In the first stage we focused on the following keywords and keyword combinations: “targeted attack {detection, identification, recognition, analysis}”; “{APT, advanced persistent threat, threat, cyber-threat}

{detection, identification, recognition}"; "semantic{s}{-}{based} {malware, intrusion} detection"; "SIEM" +"APT" as well as further combinations including the "model" or "framework" suffix.

For data providers, we disregarded all publications released prior to 2003. General papers (e.g. original publications on key topics) were not filtered by date. These initial searches yielded a total of 112 papers. Every publication was perused and summarized.

In stage 2, we assessed and selected all additional references used in the grant proposal document of the corresponding APT research project[1] at the St. Pölten University of Applied Sciences. Stage 2 increased the number of considered papers to 114.

Lastly, we extracted a sample set of relevant references from papers we identified as key publications. This increased the total number of considered works to 123. Below exclusion criteria and quality assessment ultimately brought this number down to 60.

2.3.3 Inclusion and Exclusion Criteria

We included all peer-reviewed articles released between 2003 and 2015. We specifically looked for publications that matched the following criteria:

- Frameworks, models, methods, formal descriptions and tools of host-based, network-based, and multi-source approaches to data collection and evaluation;
- Papers that explicitly focused on the detection, identification, recognition, and analysis of (targeted) attacks.

Product specification sheets, magazine articles and gray literature were excluded from our search. In addition, we removed the following material:

- Articles which focused excessively on visualization, code manipulation, honeypot technology, and social engineering;
- Papers about data stream processing techniques;
- Marketing texts and general descriptions of commercial malware analysis suites;
- Well-established industry solutions already discussed in dedicated survey papers [89, 330];
- Patents, as they typically lack in replicability and presentation.

2.3.4 Quality Assessment

The quality assessment of the reviewed articles is based on a number of questions related to both overall soundness and APT detection suitability of the presented solutions. Each quality assessment factor Qn was graded from 0 to +1. The composition of the score is detailed below:

1. Were the research questions and the overall goal clearly defined?

[1] Josef Ressel Center for Unified Threat Intelligence on Targeted Attacks

- Research question exists (+0.5)
- The overall objective and motivation is stated in accordance with the goals identified in Section 2.4.1 and Table 2.4 (+0.5)

2. Was the attack detection domain introduced and explained?
   - An attack domain (see 'Domain' in Section 2.4.1 and Table 2.4) can be clearly assigned (+0.5)
   - A practical application is exemplified (+0.5)
3. Was the approach presented in a clear and replicable manner?
   - Starting point and core process are described (+0.5)
   - Evaluation exists (+0.5)
4. Did the article explain the nature and type of operational data used in the process?
   - Base (origin) data and its representation are stated (+0.5)
   - Result (outcome) data and its representation are stated (+0.5)
5. Can the proposed solution be applied to the detection of targeted attacks?

Specifically, a paper was awarded 1 point for a category when: the research question, motivation and goals were clearly defined (Q1), the domain was well specified (Q2), the approach to solving the stated problem was presented in detail (Q3), the specifics of the input and output data were provided (Q4), and the solution is applicable to the field of targeted attacks or one of its primary aspects (Q5). Since Q5 cannot be objectively computed by evaluating definitive indicators, the score was determined in the course of an expert discussion among at least 4 security researchers per paper.

Complementing the QA score, a domain expert rating ranging from 0 to 5 was added following the initial assessment. This score is the result of a group discussion for each paper (minimum of 3 researchers, excluding the primary author) and mirrors their view on the paper independent from formal criteria. To remove any bias, this second grade was awarded without prior knowledge of the QA score.

In addition to the exclusion criteria, we used the quality assessment to reduce the number of papers ultimately included in the survey. Each article with an overall QA + expert score below 4.5 as well as papers with a Q5 score of zero were removed from the list.

2.3.5 Data Collection

The following data was extracted from each reviewed paper:

- The type of source (journal, conference, book) and full reference;
- Classification of general type of article (model, method, framework, formal definition, or tool);
- Classification of information system domain (host, network, multi-source, or general semantics);

- Classification of data gathering, analysis/detection, and learning techniques used;
- Summary including tags identifying core topics and goals;
- Quality evaluation in regards to the possible contribution to the detection of malicious, ATA-related activities or patterns.

Data extraction and initial checking was done by four security researchers.

2.3.6 Data Analysis

The extracted data was analyzed and put into tables in order to provide a quantitative overview:

- Total number of papers and papers per year, addressing R1;
- Number of papers per type of article and domain, addressing R1 and R2;
- Tags describing the key factors of the respective solution appended to each paper, addressing R2 and R3;
- Quality score of all papers as well as an APT detection applicability rating, addressing R3 as well as Q1 through Q5;
- Full table containing used technology, specific approach, technical properties and capabilities of the solutions introduced in the reviewed articles, addressing R4.

All tables as well as the statistical breakdown of the results can be found in Section 2.6.

2.4 Review Categories

We developed a versatile schema with 4 distinct categories to classify the surveyed articles. Although similar in part to Jacob et al.'s [139] behavioral malware detection taxonomy, our schema included additional properties relevant to APT detection and analysis, e.g. specific input types and general goals that aren't usually seen in malware-centric approaches. Figure 2.2 presents an overview of the properties investigated. The schema can be freely applied to all topical literature and will help categorize solution capabilities ranging from data collection and analysis to automated learning.

Primary categories, identified by an asterisk, are used for synthesizing the assessed papers and represent tags assigned to each solution. These tags include the general goal G of the solution, the type of threat T, data input type I, detection method D, and analysis technique A. Knowledge generation K is demarcated as either true or false and mentions the solution's learning and classification capabilities, where applicable. The type of semantic affinity S (see Section 2.2.2) is identified as well. This is complemented by the mapping of the technique to one or more of the APT categories (including a brief rationale) by Hutchins et al. [133] (see Section 2.2), thereby satisfying all the semantics-side selection criteria previously defined.

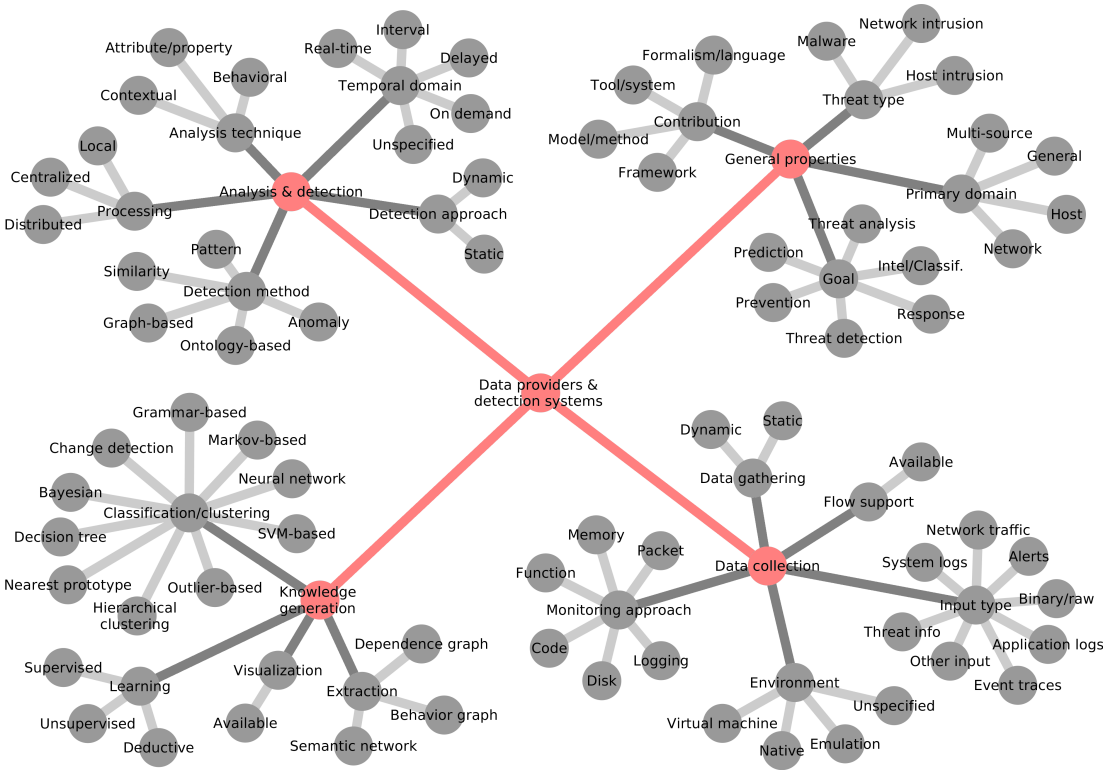


Figure 2.2: Categorization of surveyed papers

For example, a paper describing a malware detection and behavioral analysis solution based on system events and including a methodology to semantically describe correct sample execution might be tagged as *◇G{detection, analysis}; T{malware}; I{event traces}; D{pattern}; A{behavioral}; K{no}; S{operational}; APT{4,5,7: host activity}*. Such a solution would be best suited to detect local (host) events that are part of APT stage 4 (exploitation), 5 (installation), or 7 (action on target).

A full breakdown of both primary and additional properties can be found in Tables 2.4, 2.5, 2.6, and 2.7.

2.4.1 General Properties

The 'General' category encompasses high-level objectives and type definitions.

- *Primary domain* – Each paper was assigned a single domain that most closely matches the solution's area of application. Domains follow the subchapters of Section 2.5 and include host, network, multi-source, and general/unspecified solutions.
- *Contribution* – This describes the type of solution introduced in the respective paper. Frameworks, models/methods, formalisms/languages, and tools or systems are possible types of contribution. One solution may fit several categories.
- *Goal G∗* – This criterion defines the general goal the authors want to achieve. Typically, this can encompass threat prediction, prevention, threat intelligence/-classification, threat correlation/event fusion, threat detection, threat analysis,

and threat response. The introduced approaches usually have more than one goal.

- *Threat type T∗* – The threat type describes the general attack the introduced solution attempts to combat. We have separated threat types into malicious code (malware), host intrusion, and network intrusion. A system can counter a specific or several types of threat.

2.4.2 Data Collection

Here, we categorize input data specifics, data gathering, and monitoring techniques.

- *Data gathering* – Distinguishes between static and dynamic data gathering. Data gathering is concerned primarily with the method employed to collect the information from a system or application. Detection of suspicious activity or code is part of the detection approach specified below. A combination of static and dynamic data gathering is unlikely, but not impossible.
- *Monitoring approach* – There are a number of ways to dynamically monitor execution information. Among them is function monitoring, packet monitoring, code monitoring, disk monitoring, memory monitoring, and conventional logging. Specialization on one approach is likely, but not universal.
- *Data input type I** – Defines the kind of input information processed by the respective solution. This may include general threat information, system logs, application logs, network traffic, traces of system events such as API and system calls, binary code or raw data, as well as alerts generated by distributed agents or third-party software like an IDS. Many solutions support several input types.
- *Flow functionality* – Solutions with the capability to record data flows are identified here. This often includes, but is not limited to, network flow-based solutions.
- *Environment* – Depending on the system environment the respective solution can be executed on, we categorize it as running natively – or 'on-device' in case of physical appliances – or as running inside a virtualized environment such as a VM or emulator. Unspecified or environment-neutral solutions are identified as well. Tools can support or utilize several environments.

2.4.3 Analysis and Detection

In this category, we take a look at how suspicious activity or code is recognized and analyzed.

- *Detection approach* – Similar to data gathering, the detection of relevant information can either happen statically or dynamically. This may in some cases differ from data gathering when the process of collection and detection is realized separately. For example, data may be dynamically recorded and subsequently undergo static analysis.

- *Detection method D** – This incorporates general detection methods such as anomaly detection, pattern/misuse detection, similarity detection (e.g. string similarity), graph matching, or ontology-based methodologies. Systems such as pure correlation solutions do not necessarily utilize a particular detection method. Several methods may be employed.
- *Analysis technique A** – Including but not limited to malware, the analysis of data and, by extension, the decision about its relevant properties, can usually happen based on pre-defined attributes (properties, states), through behavioral analysis techniques, or at a contextual (semantic) level. In most cases, solutions use only one approach.
- *Temporal domain* – Depending on their implementation, detection or analysis solutions can offer real-time, delayed but continuous, fixed-interval, or on-demand processing initiated by user command or upon detection of a certain event.
- *Processing* – Data processing itself can be performed locally on the system the data is stored/collected, in a centralized fashion on e.g. a dedicated server, or distributed across several nodes.

2.4.4 Knowledge Generation

A number of solutions offer mechanisms to generate knowledge from the collected and analyzed data.

- *Learning* – (Machine) learning capabilities are identified here. Supervised learning uses available malicious and benign data to determine the difference to a baseline, unsupervised learning tasks the system to determine the deviation by itself, and deductive learning uses inference to draw conclusions based on known premises.
- *Classification and clustering* – This category lists all the identified techniques used to classify or cluster information. We consider support vector machines (SVM), decision trees, neural networks, methods based on nearest-neighbor or nearest-prototype determination, Bayesian techniques and respective statistical, 'belief'-based systems, Markov models or other 'memoryless' statistical models, grammars realized through e.g. grammar parsing, outlier detection, change detection such as state comparison, and hierarchical clustering.
- *Extraction* – In some cases, data is merely extracted for later analysis or visualization. Techniques found in the reviewed papers include dependence graphs, behavior graphs, and semantic (link) networks.
- *Visualization* – As a by-product of knowledge generation, the results of earlier assessment stages can be graphically presented. Solutions offering visualization are identified here. See [330] for further details about security-related visualization systems.

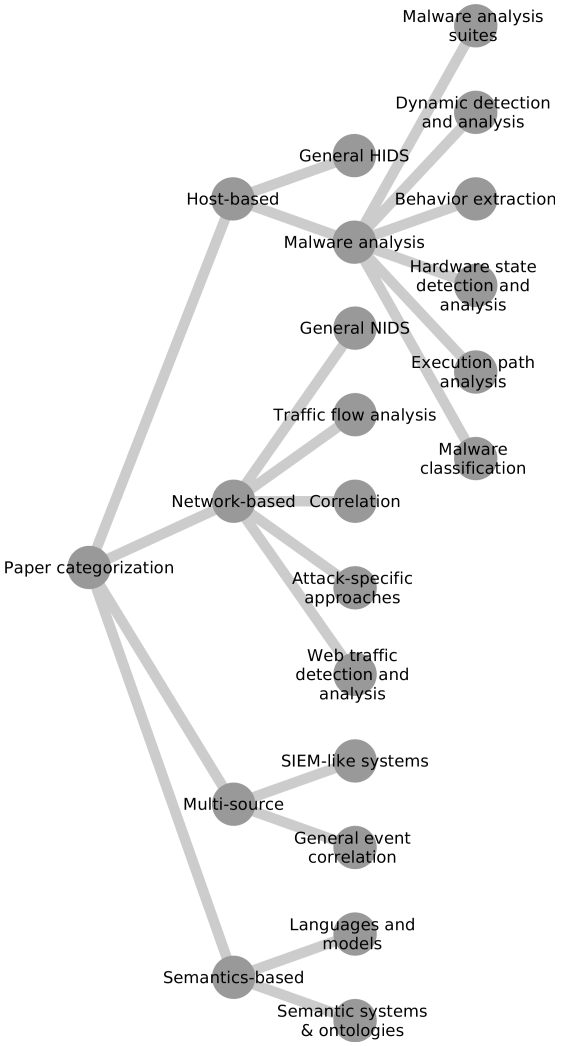


Figure 2.3: Paper categorization

2.5 Review

In this section we assess and summarize all the papers found through the structured search process specified above. Solutions are categorized into host-based, network-based, multi-source, and purely semantic approaches that cannot be attributed to a specific domain. Figure 2.3 offers an overview of the full categorization used in below subsections.

The beginning of each category section provides some additional background relevant to the respective category. Tags are used to identify primary categories for each paper, providing information about the assessed solution's general approach to data gathering, monitoring, detection and analysis, as well as knowledge generation.

2.5.1 Host Domain

Host-based solutions can be understood as detection systems running on the endpoint. We identified memory-based approaches, numerous behavioral detection and analysis

systems, function call monitoring solutions, host-based intrusion detection systems, and more. For many of the tools, *malicious software* (malware) is something of a common denominator:

Most cyber-attacks involve malware smuggled onto the system to perform its sinister deed. Malware can generally be defined as *"any software that does something that causes harm to a user, computer, or network"* [291]. Examples include viruses, Trojan horses, backdoors, worms, rootkits, scareware, or spyware. Malicious software is known to exploit vulnerabilities of the system it is designed to run on. Flaws in applications can serve as drop vector and may be exploited as part of a privilege escalation routine required for administrative tasks. In the case of APTs, this often includes hitherto unknown exploits known as zero-days.

Common malware i.e. aims at financial gain through fraud or blackmail; it is delivered to a large number of recipients in hope that a sufficient number of people unknowingly install it on their machines. Malware used for targeted attacks is much more sophisticated and typically includes additional components for e.g. long-term persistent installation and more complex command and control communication than can be found in other malicious software. The main distinguishing factor is its tailoring to a specific environment, however. APT malware is designed to attack a particular device, operating system, or specific application version. This influences the choice of dropping technique, evasion routines, and attacks against specific defensive measures employed by the victim. While this fact makes such software dangerous to only a limited number of systems, the damage caused by targeted malware can be much more severe.

Malware analysis is probably the most widely used and arguably most important class of host-based approaches. Solutions range from analysis suites built to determine a sample's general maliciousness to interpretation of behavior or clustering of potential malware into families. [89] and [134] offer an overview of dynamic malware analysis tools and describe various monitoring and detection techniques. In a previous paper [330], we expanded on some of the tools and assessed the information they provide. There is a multitude of information to be gleaned from various malware analysis techniques, each offering specific insight into the nature and functionality of a malicious program. This includes the virus definition, packer information like packer designations and general compression information about the sample, file and header information and code sections, library and function imports, CPU instructions and their associated assembly operations, system and API calls executed by the sample, file system operations indicating the creation, modification, and deletion of files, as well as interpreted registry, process, thread, and network operations.

A large number of the data providers surveyed below focus on the detection or analysis of malware. We generally define malware data providers as tools that utilize static or dynamic analysis methods as well as signature- and behavior based detection techniques to gather information about a potentially malicious piece of software [330]:

Static analysis describes techniques that do not require the sample under scrutiny to be actually executed. Depending on the depth of analysis, a file may be checked for its basic properties like file type, checksum, easily extractable information such as null-terminated strings or DLL import information, or be fully disassembled [156]. The analysis environment – bare metal, virtual machine, or emulation – plays a negligible role for static analyses – the analyst simply chooses a platform compatible with the tools of her choice. *Dynamic analysis* goes a step further and executes the file on a dedicated host system. Various tools then monitor the execution and log relevant information into an execution trace. This ranges from simple file system operations to a full instruction list captured through a debugger. The analysis environment is essential for the dynamic approach since the type of data logged depends on both the platform as well as on the techniques used to capture system events.

On the detection side we differentiate between signature-based and behavior-based techniques:

Signature-based approaches are best known for their prominent role in antivirus software and traditional intrusion detection systems. A so-called definition or signature is created to describe an entire file or parts of the code that are known to be malicious [330]. The detection software then compares the appearance of a file or packet to this set of known signatures. Signature-based detection has several shortcomings [80]:

Firstly, obfuscation techniques commonly utilize polymorphic or metamorphic mutation to generate an ever-growing number of malware variants that are different in appearance, but functionally identical. This leads to bloated signature databases and, ultimately, to an overall slowdown of the detection process. Secondly, signature-based techniques only detect malware which has already been identified and analyzed; new species or hitherto unknown variants are often overlooked.

Behavior-based techniques, on the other hand, focus on specific system activities or software behavior typically captured through dynamic analysis. Malicious actions are defined through patterns or behavioral anomalies. Since the behavior-based approach can be semantics-aware, it is largely immune to obfuscation [80]. Its performance is limited, however: While signature matching takes but the fraction of a second, dynamic execution and trace analysis can take several minutes.

The second big area of host-based threat mitigation is *intrusion detection*. IT systems intrusion describes the act of accessing a local or network-based resource without proper authorization. It can basically be defined as computer break-in leaving the targeted system vulnerable to theft or sabotage [170]. In many cases the line between black-hat hacking and malicious software becomes blurred [150]. Most typical intrusions involve malware or tools designed to circumvent security measures of the target system. The attack procedure varies from case to case and can be considered 'targeted'; few intruders act without knowing which system they are attacking.

On the defense side, intrusion detection is primarily concerned with activities deemed harmful by the system operator [170]. These may include digital breaking and entering but generally encompasses all 'malicious actions' aimed at e.g. privilege acquisition or resource abuse. Literature differentiates two classes of intrusion detection systems (IDS): *network-based* and *host-based* [329]. The main differences are the type and number of sources used to detect adversary activity; network-based IDS analyze the traffic transmitted between systems and typically utilize a number of sensors distributed throughout the network. Host-based systems, on the other hand, attempt to discover attacks taking place on the very machine the sensor component is installed on.

There are two approaches to detecting malicious activity through an IDS: Anomaly detection and misuse (pattern) detection [170]. *Anomaly detection* is based on the premise that illegal activity manifests as abnormality and that it can be identified through measuring the variance of certain key metrics. This may include the excessive use of system functions, high resource utilization at unusual times, or other behavior deviating from a defined baseline.

Misuse detection, on the other hand, is based on predefined patterns. Knowledge of an attacker's methods or the expected consequences of an attack are encoded into behavior sequences that can be found by an IDS looking for their occurrence [170]. Examples include sequences of suspicious function calls, certain network packet payloads, or the exploitation of known bugs.

Many of the reviewed papers describe various host-based approaches to locate, identify, categorize, or analyze malicious software. Both semantics-aware as well as a few semantics-based techniques are used. Most focus on determining the maliciousness of a piece of software or attempt to classify a set of samples.

There are also various data providers that fall under the wider definition of an intrusion detection system. Since IDSs can be both host- and network-based, tools range from local process or system call tracers to traffic anomaly detectors utilizing various statistical methods or data mining approaches. Network-based systems are reviewed in Section 2.5.2.

Malware Analysis Solutions

Inspired by [139], we categorized solutions into malware analysis suites, dynamic and static detection and analysis solutions, hardware state detection and analysis, execution path analysis, behavior extraction systems, as well as malware classification systems.

Malware analysis suites. While malware analysis suites are mostly established commercial solutions and thereby not included as per the criteria defined in Section 2.3, they are an important class of data providers that need to be mentioned. Malware suites do not per se run on the endpoint: Tools like Anubis/LastLine Analyst [24, 137], Cuckoo Sandbox [67], CWSandbox/Threat Analyzer [343, 342], Joe Sandbox [144], and FireEye MAS [100] run in their own, usually virtualized environment. They employ

function hooking and kernel mode drivers to record and report system and/or API calls executed by the sample under scrutiny. The recorded activity is then interpreted and returned as human-readable summary. Malware analysis suites are a good starting point for general dynamic analysis and are undoubtedly the inspiration for some of the solutions introduced below. See [89] and [330] for more information. ⋄*Representative tags: G{detection, analysis}; T{malware}; I{event traces, network traffic}; D{pattern}; A{behavioral}; K{no}; S{aware}; APT{3-7: host and network activity}.*

Dynamic malware detection and analysis is covered by a multitude of articles: Grégio et al. [120] introduce BehEMOT, a non-intrusive malware behavior analysis system. It uses both a native and emulated system environment that promises to circumvent sandbox detection. Functions and their arguments as well as network events are captured using system call interception via System Service Dispatch Table (SSDT) hooking: The SSDT is a list of memory addresses that each correspond to a system call. If hooked, it becomes possible to redirect calls to an altered copy of the function; a technique useful for monitoring activity or modifying results. Similar to commercial suites like Anubis [137] or Joe Sandbox [144], BehEMOT abstracts specific calls to a general operation type (e.g. 'open process'). ⋄*Tags: G{detection, analysis}; T{malware}; I{event traces, network traffic}; D{pattern}; A{behavioral}; K{no}; S{aware}; APT{3-7: host and network activity}.*

Fukushima et al. [108] developed another behavior-based detection approach: Suspicious process behavior is identified through specific system and temporary directories accessed by the sample as well as the creation of certain registry keys responsible for e.g. automated startup. Registration or deletion of uninstall information is considered as well. Fukushima's system utilizes Procmon [270] as primary data provider. Unlike BehEMOT's post-monitoring abstraction [120], Procmon abstracts system calls in a non-transparent manner prior to analysis. For this reason, many additional calls are not considered. This arguably increases performance while potentially reducing accuracy. ⋄*Tags: G{detection, analysis}; T{malware, host intrusion}; I{event traces}; D{pattern}; A{behavioral}; K{no}; S{aware}; APT{4,5,7: host activity}.*

In [132], the authors introduce the Taiwan Malware Analysis Net (TWMAN), an ontology system for behavioral malware analysis. The VM-based dynamic analysis system is built around the TRUMAN sandnet [75] that logs contacted IP addresses, created, changed, and deleted files, as well as registry activity. While some features are taken from CWSandbox [342], others are comparable to the generally more detailed BehEMOT approach [120]. The file output is akin to a list of changes to the original system state. An OWL-based ontology built in Protégé [296] describes the impact and approximate activity of each sample and includes a rough categorization into different types of malware. Actors, assets, and activity beyond abstracted malware function calls are not considered in the proposed ontology. ⋄*Tags: G{intelligence, analysis}; T{malware}; I{event traces}; D{ontology}; A{behavioral, contextual}; K{no}; S{operational}; APT{3-7: host and partial network activity}.*

The work of Chiang and Tsaur [50] introduces dedicated, ontology-based behavioral analysis for mobile malware. Their approach is based on the TOVE project by Fox et al. [104] and considers infection routes, potential damage, and propagation capabilities via e.g. Bluetooth. While the authors do not directly propose a practical implementation, their work can be seen as example of a domain-specific, ontology-based approach that could be transported to the information system environment. ⋄*Tags: G{analysis}; T{malware}; I{-}; D{ontology}; A{bahavioral, contextual}; K{no}; S{denotational}; APT{-: general approach}.*

Static malware detection and analysis is less concerned with application behavior, but primarily considers a sample's code structure. Zhang et al. [356] introduce a detection technique for polymorphic code through static identification of its self-decryption functionality. Limited emulation of instructions is used in conjunction with recursive traversal of loops to find the starting location of the respective routine. Since Zhang et al. address a very specific property of only a subclass of malicious programs, their approach could potentially complement systems that attempt to identify other function entry points [83] or illegitimate code paths [37]. ⋄*Tags: G{detection}; T{malware, host intrusion}; I{network traffic, raw}; D{pattern}; A{behavioral}; K{no}; S{aware}; APT{3,5: partial network and initial program activity}.*

Dube et al. [83] take a more general approach. Their system, MaTR, focuses on target recognition of malware attacks through static heuristic features. These non-instruction-based heuristics include structural anomalies such as non-standard section names, characteristics such as the presence of unpacking routines, entry points outside the code section, and unusual numbers of function imports and exports. MaTR utilizes supervised machine learning based on bagged decision trees. Dube's research contrasts the n-gram approach found in other surveyed articles [263, 47, 32], where sequences of byte patterns of various lengths are assessed. ⋄*Tags: G{detection, analysis}; T{malware}; I{raw}; D{anomaly}; A{behavioral}; K{yes: supervised learning, decision tree classification}; S{aware}; APT{5: structural program properties}.*

One of the most cited works is undoubtedly by Christodorescu et al [53]. The authors describe a malware detection algorithm that incorporates instruction semantics, an approach that uses resilient pattern matching which can deal with slight variations in code. Activities such as decryption loops and search operations for certain strings are considered malicious behavior indicators, which are specified by templates and depicted as control-flow graphs (CFG). Christodorescu et al. also developed a tool based on IDA Pro [127]: It takes a binary, disassembles it, constructs a CFG, produces an intermediate form using abstract machine language, and determines a possible match. In a later paper, Preda et al. [69] prove that the concept of semantic malware detectors such as [53] is a sound one and can indeed defeat numerous obfuscation techniques. ⋄*Tags: G{detection, analysis}; T{malware}; I{raw}; D{pattern}; A{behavioral, contextual}; K{no}; S{aware}; APT{4,5,7: program activity}.*

Sharif et al. [287] introduce Eureka, a framework for facilitating static analysis of obfuscated code. The sample is first executed to trigger its unpacking procedures,

then captured and dumped for static analysis. Here, API resolution comes into play: Eureka identifies subroutines and builds a control flow graph for every function. Calls are then extracted from the image of the previously packed sample. The combination of static and dynamic analysis addresses the issue that packed or encrypted samples are hard to analyze in their original state. Letting the respective routines execute before taking a closer look on the disassembled code promises an unimpeded view on the malware in question – with any of the above-mentioned static analysis tools. ◇*Tags: G{analysis}; T{malware}; I{event traces, raw}; D{pattern, ontology}; A{attribute, behavioral}; K{yes: behavior graph extraction, visualization}; S{aware}; APT{5: initial program activity}.*

Hardware state detection and analysis solutions encompass raw operation monitoring applications that are often VM-centric. Fore example, the approach described in [143] focuses specifically on virtual machines. VMwatcher constructs an on-system semantic view on files, processes, and kernel modules to bridge the gap between the inside and outside view on a system. Jiang et al. use virtual machine introspection (VMI) to monitor the raw disk and memory states of a VM in a transparent and tamper-resistant fashion. This ultimately eliminates the need to install malware detection software on the guest machine and could enable further research into system state analysis in virtual environments. ◇*Tags: G{detection}; T{malware}; I{event traces, raw}; D{-}; A{attribute}; K{no}; S{aware}; APT{4,5,7: raw host activity}.*

Payne et al. [243] also focus on VMI-based monitoring. Their proposed solution, Lares, is primarily intended to protect other defense measures such as antivirus programs and intrusion detection systems by providing an isolated API monitoring environment. Lares could be combined with other tools that would benefit from the additional protection against evasion or tampering. ◇*Tags: G{detection}; T{malware, host intrusion}; I{event traces, raw}; D{-}; A{-}; K{no}; S{aware}; APT{4,5: raw host activity, secondary protection}.*

Mankin and Kaeli [199] propose a similar disk monitoring system: Their implementation, the Disk I/O analysis engine (DIONE), intercepts and interprets disk access operations using a sensor that resides in the Xen hypervisor outside the guest OS. Thanks to this design, DIONE is more resilient against many conventional attacks and various obfuscation techniques. Attacks against the hypervisor itself [244] and cross-VM attacks [265, 17] remain an issue, however. Unlike Jiang's VMwatcher [143], DIONE is capable of monitoring the NTFS filesystem. ◇*Tags: G{detection}; T{malware}; I{raw}; D{-}; A{-}; K{no}; S{aware}; APT{4,5: raw disk operations, secondary protection}.*

Execution path analysis explores or defines code paths that are dynamically investigated during execution. Miles et al. [213] focus on the interrelationship among malware instances to discover new connections between actors, machines, and malware: Code, semantically similar procedures of code, and API call execution/event log traces are compared to identify similarities. This includes websites, e-mail messages and PE file headers. Names, addresses, and certain constants are generalized using BinJuice [172]. The discovery system itself uses concolic execution: It is able to explore multiple exe-

cution paths by using an SMT solver to create new input values for subsequent runs. Miles' prototype system, VirusBattle, further includes an unpacking routine as well as monitoring capabilities utilizing VMI. With a focus on threat intelligence and attribution, Miles' approach could be combined with other, more detection-centric methods to enhance its capabilities. ◇*Tags: G{intelligence, analysis}; T{malware}; I{event traces, raw}; D{similarity}; A{attribute, contextual}; K{no}; S{operational}; APT{3-7: host and partial network activity, web, e-mail}.*

Another interesting approach is presented by Xu et al. [347]. They introduce a waypoint mechanism in the form of markers on the execution path that a process must follow to provide trustworthy control flow information for subsequent anomaly monitoring. Waypoints, for which Xu et al. use the context of the currently active function of the code, are generated through static analysis and later imported as traps into the kernel. A system utilizing this method would be able to detect mimicry attacks [329], i.e. the interleaving of malicious calls in benign sequences. ◇*Tags: G{detection}; T{malware, host intrusion}; I{raw}; D{anomaly}; A{contextual}; K{no}; S{operational}; APT{4,5: program flow}.*

PECAN [37] is a dynamic anomaly detector that identifies unusual program behavior through the definition of legitimate code paths. It has been developed to identify bugs that use valid executions but violate the programmer's expectations. The authors introduce both a training and monitoring module: The system scans for specific security functions that can affect the system outside the Java VM, considers the calling context of a call, and checks for anomalous sequences. This is achieved through probabilistic calling context (PCC), which appends a unique number that represents the location a program is called from [36]. Clients then query that number at every system call to determine whether the context is known or suspicious. While PECAN is a solution specific to Java, the concept of context can also be found in system calls [166]. Synergies to process-graph-based systems are likely, but have yet to be investigated. ◇*Tags: G{detection}; T{malware}; I{raw}; D{anomaly}; A{bahavioral, contextual}; K{supervised learning}; S{operational}; APT{4,5: program flow}.*

Behavior extraction solutions often focus on graphs and subgraphs of certain behavioral commonalities. Often similar to classification approaches (see below), they usually focus on the generation of knowledge.

Kwon and Lee [171] introduce BinGraph, a discovery method for metamorphic malware. API calls are extracted and converted to a hierarchical behavior graph. Extracted subgraphs represent common behavior and can be considered a 'semantic signature' – Kwon and Lee assume that the same API sequences occur in metamorphic variants of the same malware strain. To counter isomorphism, node types are abstracted into categories such as process, registry, memory, or socket operations. Each operation can be further split into groups that denote the specific action (open, close, read, write) performed. Graphs are matched to search for subgraphs in newly submitted traces. ◇*Tags: G{detection, analysis}; T{malware}; I{event traces}; D{graph}; A{attribute}; K{yes: behavior graph extraction}; S{aware}; APT{4-7: host activity, sockets}.*

In [334], the Wang et al. describe a malware feature extraction system based on application behavior. System calls and their dependencies are mapped to a graph and then tainted to trace the passing on of parameters. Calls of interest include file, registry, process, and network functions. Single-step debugging is used to keep track of the tainted information. Semantic analysis is used on certain, not further specified calls. The taint approach sets this work apart from other solutions and allows the authors to extract calls that are actively used by malware. ⋄*Tags: G{intelligence, analysis}; T{malware}; I{event traces}; D{graph}; A{behavioral, contextual}; K{yes: dependence graph extraction}; S{aware}; APT{4-7: host and partial network activity}.*

Another graph-based approach, DAVAST, is described by Wüchner et al. [346]: The authors visualize trace information as quantitative data flow graphs where files, sockets, processes, and more are represented as nodes while the edges depict the execution flow. Both online and offline analysis is supported. Rules define normal behavior as well as malicious activities. Abstraction of e.g. memory addresses is used to reduce processing complexity. Unlike most other solutions, Wüchner et al. use time slices to further reduce the size of their graphs. Since DAVAST primarily focuses on visualization, further automated processing it is not explored. It is currently unknown if Wüchner's approach [346] could be used in combination with e.g. Wang's feature extraction system [334] or other graph-based approaches such as BinGraph [171] or the system introduced by Dolgikh [79]. ⋄*Tags: G{detection, analysis}; T{malware, host intrusion}; I{event traces}; D{pattern}; A{behavioral}; K{yes: visualization}; S{aware}; APT{3-7: host activity, sockets, e-mails}.*

Dolgikh et al. [79] conduct behavioral analysis capable to automatically create application profiles for both malicious and benign samples. Their system considers recorded API calls that are subsequently transformed into a labeled graph representing a stream of system calls. Graphs are compressed using the Graphitour algorithm [245] used in genetic data processing. When applied to benign software, the resulting dataset describes rules that represent normalcy – functionality of system calls that can be seen as benign behavior. Ultimately, each node in the rule graph is modeled as part of a Colored Petri Net [142]. ⋄*Tags: G{intelligence, detection}; T{malware}; I{event traces}; D{anomaly, graph}; A{behavioral}; K{yes: supervised learning, classification grammar, behavior graph extraction}; S{aware}; APT{4,5,7: host activity}.*

Graph mining is another interesting approach to extracting behavior. Next to developing a language for specifying malicious behavior through system call dependencies, Christodorescu et al. [54] came up with a method to mine behavioral data from trace files of benign and malicious samples. Both are collected during dynamic analysis and linked through their parameters. Christodorescu's system, dubbed 'MiniMal', constructs dependence graphs for later comparison. To keep the classification process non-specific to individual samples, the graphs are generalized into so-called minimal contrast subgraphs describing the smallest possible malicious subgraph that does not occur in benign sequences. Other works build on Christodorescu et al.'s concept: Fredrikson et al. [106] introduce a graph mining method for extracting significant behavioral

specifications used to describe classes of programs. Their system, Holmes, aims to extract specifications that distinguish malicious from benign applications on a system call level. ⋄*Tags: G{intelligence, detection}; T{malware}; I{event traces}; D{graph}; A{behavioral}; K{yes: deductive learning, dependence graph extraction, visualization}; S{aware, denotational}; APT{4,5,7: host activity}.*

Touching the network domain, Jacobs et al. [141] present Jackstraws, a system designed to identify command and control (C2) communication. Unlike other network-centric approaches, Jackstraws captures host activity through dynamic analysis performed by tools such as Anubis [137] and associates network communication to local malware activity. Association is achieved through behavior graph modeling of data flows between individual system calls. Graph templates for C2 pattern similarity matching are mined from a known set based on a technique introduced by Yan and Han [350], followed by a clustering stage. While Jackstraws is potentially vulnerable to certain obfuscation and mimicry attacks [329], its unique approach to detecting C2 traffic makes it particular interesting to APT detection and knowledge generation efforts. ⋄*Tags: G{intelligence, detection}; T{malware}; I{event traces}; D{pattern, similarity, graph}; A{behavioral, contextual}; K{yes: supervised learning, graph clustering, behavior graph extraction}; S{aware}; APT{6: C2 network activity}.*

Malware classification solutions found in literature usually cluster dynamic analysis traces or static function use. The primary purpose of malware classification is the generation of knowledge through sample similarity assessment. Unlike behavior extraction (see above), classification does not necessarily yield discriminative patterns for subsequent use.

Riek et al. [314, 263] describe a clustering and classification approach for malware behavior traces generated by dedicated dynamic analysis tools (in their case the malware sandbox 'CWSandbox' [342]). The focus lies on API and system calls as well as their arguments. The reports are embedded in a vector space in order to enable similarity assessment based on geometry. Hierarchical clustering is then used to identify groups of malware displaying similar behavior. The distance to a representative prototype is determined and stored. Incremental analysis allows for comparing new samples to existing clusters. For better granularity, the authors introduced the optional Malware Instruction Set (MIST) format: The behavior of a binary is described as a sequence of instructions similar to CPU opcodes. Each level of MIST allows for additional details to be included or omitted on demand. Riek et al.'s work later became 'Malheur' [262], a system capable of rapidly processing function call n-grams. Malheur's performance and flexibility makes it a useful tool for pre-classifying samples without having to convert existing traces. However, since Riek's approach does not extract patterns like e.g. Christodorescu's work [54], Malheur's performance and accuracy comes at the expense of semantic expressiveness. ⋄*Tags: G{intelligence, detection}; T{malware}; I{event traces}; D{-}; A{behavioral}; K{yes: unsupervised learning, nearest prototype clustering}; S{aware}; APT{4-7: support solution}.*

In [25], Bayer et al. present a scalable behavior-based malware clustering system. Dynamic analysis is performed using Anubis [137] and extended with taint tracking functionality that marks out-arguments and return values and monitors them for changes. The resulting trace files are then generalized into profiles which characterize sample behavior. Ultimately, the abstracted data is clustered using the locality sensitive hashing (LSH) algorithm based on the work of Indyk and Motwani [135]. Despite the fact that Riek's approach is more accurate in terms of F-measure [263], the extraction of profiles is a distinct advantage over pure classification systems like Malheur. ◇*Tags: G{intelligence, detection}; T{malware}; I{event traces}; D{pattern, similarity}; A{behavioral}; K{yes: nearest prototype classification}; S{aware}; APT{4-7: host and limited network activity}.*

Host-based Intrusion Detection Systems

Host-based intrusion detection systems focus less on the tool/malware utilized by the attacker but rather on anomalous activities registered on the system. For example, Mutz et al. [166] describe a classical approach to detecting anomalies in system call sequences. Their system is designed to detect attacks against privileged applications. To this end, it analyzes the relation between system call arguments and calling contexts. Function return addresses gathered from the application call stack are used to add cohesion. Among the anomalies considered are string length, character distribution, and the use of certain non-printable characters. As part of Mutz' prototype implementation, a Linux kernel module monitors system calls through a wrapper function that logs relevant activity before actual execution. ◇*Tags: G{detection}; T{host intrusion}; I{event traces, raw}; D{anomaly}; A{behavioral, contextual}; K{yes: supervised learning, Markov-based classification}; S{aware}; APT{4,5,7: host activity}.*

Anomalies in system call patterns are a predominant theme: Creech and Hu [66] introduce a host-based anomaly detection method that uses discontiguous (unconnected) system call patterns. A context-free grammar describes benign and malicious call traces; learning is achieved through an Extreme Learning Machine (ELM), a new type of fast-learning artificial neural network. Several decision engines were tested and compared by the authors, making the paper a good starting point for the selection of learning algorithms applicable to system call sequences. ◇*Tags: G{intelligence, detection}; T{host intrusion}; I{event traces}; D{anomaly}; A{contextual}; K{yes: supervised learning, neural network/Markov-/grammar-based classification}; S{aware w/ denotational elements}; APT{4,5,7: host activity}.*

A way to formally describe a data flow without losing information about the accessing resource is described in Chaturvedi's work [47]. Their model can capture certain properties of the sample and identifies anomalies hinting at malicious use of functions or configuration files. Among the considered properties are command line arguments and environment specifics such as local variables. ◇*Tags: G{detection}; T{malware, host*

intrusion}; I{event traces}; D{anomaly}; A{behavioral}; K{yes: supervised learning, behavior graph extraction}; S{aware}; APT{4,5,7: host activity}.

Based on this approach, Bhatkar et al. [32] propose anomaly detection within the data flow of an application: In addition to calls, call arguments are considered and used to establish a temporal context. Unary relations define the properties of an argument in the form of a specific value or range, while binary relations establish the relationship between two event arguments. This model allows to search for e.g. certain paths or file extensions and is able to compare them to each other. With its focus on learning as well as matching control flows, both Bhatkar's [32] and Chaturvedi's [47] systems exemplify and discuss the use of n-gram and execution graph methods. ⋄*Tags: G{detection}; T{malware, host intrusion}; I{event traces}; D{pattern, ontology}; A{contextual}; K{yes: supervised learning, Markov-based extraction}; S{aware, denotational}; APT{4,5,7: host activity}.*

Ou et al. [238] emphasize the issue that attacker and user actions are often syntactically similar and are therefore hard to distinguish from one another. They propose the formal modeling of uncertainties to counter false-positives and use various system monitoring data sources to score them. The introduced reasoning engine is designed to determine whether a system is actually compromised. Statements written in the logic programming language Prolog [61] describe the rules and aim to emulate the reasoning process of a human administrator. ⋄*Tags: G{prediction, detection}; T{host intrusion}; I{system/app logs, network traffic, event traces}; D{-}; A{-}; K{no}; S{denotational}; APT{3-7: general approach}.*

Kumar and Spafford [170] take a step back and present a pattern matching model for misuse intrusion detection, which is not based on the premise that intrusive activity always manifests as an abnormality. Instead, misuse detection uses knowledge about attacks such as known exploits as well as patterns and monitors the system for the occurrence of these patterns. The approach proposed in their paper defines the patterns as state transition graphs that are an adaptation of Colored Petri Nets [142]. Start and final states as well as the paths in between are matched by the net. Like most pattern-based approaches, Kumar's system only looks for known behavior. ⋄*Tags: G{detection}; T{host intrusion}; I{-}; D{pattern}; A{behavioral}; K{no}; S{aware}; APT{4,5,7: general approach}.*

Andersson et al. [13] focus on code injection that can be mapped to DLL injection attacks. Their framework uses process tracing and DLL hooking based on the Microsoft Detours library: Potentially executable instructions are identified through Snort [56] and Fnord [44] and are sent to a monitored environment where they are dynamically scanned for shellcode. Two methods were employed by the authors: NOP detection similar to [53], and executable code identification. Andersson's framework is an example for how suspicious traffic data can be automatically forwarded by an IDS to a dynamic host-based analysis tool. ⋄*Tags: G{detection, analysis}; T{host intrusion}; I{network traffic, event traces}; D{pattern}; A{behavioral}; K{no}; S{aware}; APT{4-7: partial host and network activity}.*

2.5.2 Network Domain

Network-based approaches primarily include generic network-based intrusion detection systems (NIDS), specific attack detection systems, traffic flow analysis solutions as well as detection systems for malicious web traffic.

Although usually classified as NIDS (see above), network traffic analysis systems deserve special attention. They come in two distinct flavors [295]: *Packet inspectors* that analyze the payload of certain (or all) packets, and *flow-based detection systems*. The latter focus on communication patterns instead of individual packets. Such patterns typically include source and destination IP addresses, port numbers, timestamps and transmission duration, as well as the amount of data and number of packets sent. Flow monitoring systems like compatible IOS routers usually export Netflow [55] or IPFIX [313] records and transmit them to a central node for analysis.

Detailed packet inspection requires significant processing capacities and may cause slowdowns on even the most powerful networks. For this reason, many solutions focus primarily on header information and analyze only specific packets such as HTTP requests and responses relevant for malicious web traffic detection. Flow-based systems are generally faster but ignore packet payloads. In addition, they are reliant on external analysis systems that amalgamate and interpret the collected data.

Both approaches may utilize pattern- and anomaly-based detection (see HIDS above). For example, a large number of SSH connections to a specific destination IP address could match the pattern of a brute-force attack. Significant amounts of data sent during the out-of-hours period, on the other hand, could constitute a data leakage anomaly.

A growing number of data providers use flow-based detection to complement classical packet inspection. It is often used to identify Denial of Service (DoS) attacks and other suspicious communication patterns. Packet scanning, on the other hand, aims at finding potentially malicious payload data. This makes it the direct equivalent to signature-based malware detection.

Many network-based systems use publicly available datasets to test the capabilities of an NIDS. Examples include the 2000 DARPA set [218] and KDD 1999 [318]. On the payload side, ADMmutate and the Clet engine [148] are often used to generate (polymorphic) shellcode.

Network-based Intrusion Detection Systems

General NIDS are represented in a large number of papers and products. Several of those include a semantic component. While commercial solutions are exempt as per the criteria defined in Section 2.3, there are a number of established (open-source) NIDS/IPS that need to be mentioned. Snort [56], Suricata [103], and Zeek [242] (formerly Bro) are among the best known projects in the security community and are utilized by some of

the solutions discussed in this chapter. Feature lists and in-depth comparisons can be found in the literature [42, 210, 241, 311, 340] and will not be reiterated here. Summed up, these products primarily offer pattern-based detection that sometimes extends to more complex rules that describe specific attacks[1].

Abdoli and Kahani [1] describe a distributed IDS that can extract semantic relations between attacks using an ontology based on the Semantic Web [328], a collection of W3C formats for data exchange. Like similar solutions [132, 50], they utilize Protégé [296] for ontology design and SPARQL [327] for querying. Their system uses Java-based JENA [16] agents to collect IPs, ports, protocols, and connection status information. A dedicated master node interprets the data and attempts to find indicators for embedded malware, buffer overflows, password attacks, or ongoing DoS activity. ◇*Tags: G{detection}; T{malware, host intrusion, network intrusion}; I{network traffic}; D{pattern, ontology}; A{attribute, contextual}; K{no}; S{aware}; APT{1,3-7: network activity with payload detection}.*

Many papers are based on or inspired by Christodorescu et al.'s work ([53], see 'Static malware detection and analysis): In [275], the authors describe two sliding-window-based schemes used to automatically generate malware signatures: a fixed-size scheme and a more flexible variable length scheme which stops at certain, pre-defined patterns. In combination, they augment Christodorescu's approach with a traffic classifier and a binary detection and extraction module capturing polymorphic shellcode. Like a part of Andersson's less successful host-based approach [13], their work focuses on the presence of 'no operation' (NOP) and NOP-like instructions. ◇*Tags: G{correlation, detection}; T{malware, host intrusion, network intrusion}; I{network traffic}; D{pattern}; A{attribute, contextual}; K{no}; S{aware}; APT{3,4,6: network activity with payload detection}.*

Hirono et al. [128] present another, more architecture-centered approach. They propose a distributed IDS that uses a transparent proxy able to analyze internal network traffic in an isolated environment. The system primarily uses signature-based detection to identify malware propagation, spamming, or DoS attacks. Suspicious binaries are automatically extracted and forwarded to a dynamic malware analysis sandbox. The decision whether a sample is suspicious is made by Snort [56] complemented by an off-the-shelf virus scanner. Similarly, Andersson [13] uses the Snort IDS for initial decision-making. Unlike Andersson and Scheirer [275], Hirono et al. do not attemt to identify shellcode but rather focus on the extraction and analysis process of the flagged binaries. ◇*Tags: G{detection, analysis}; T{malware, host intrusion, network intrusion}; I{network traffic, event traces}; D{pattern}; A{attribute}; K{no}; S{aware}; APT{3,7: network activity with normal malware scan}.*

Correlation of intrusion events is a vital part of most multi-agent IDS systems. Chien et al. [52] introduce a primitive-attack (PA)-based correlation framework able to detect multi-stage attacks. IDS alerts from various tools such as Snort [56] are ex-

[1] https://www.snort.org/rules_explanation

tracted, stored and evaluated. Through time window correlation complemented by port and IP matching, the system is able to identify e.g. network scans and denial of service attacks. The authors construct attack templates, mapping abstracted goals to specific attacker actions such as reconnaissance, penetration, and other unauthorized activity. ⋄*Tags: G{correlation, detection}; T{network intrusion}; I{alerts}; D{ontology}; A{contextual}; K{no}; S{axiomatic}; APT{1,3,6,7: partial network activity}.*

Zhu and Ghorbani [358] present an alert correlation technique that also considers attacker strategies. Their classification encompasses a neural network as well as a support-vector machine (SVM) [64] approach. Their system suggests relationship of alerts: For example, consequences of one event are mapped to the prerequisites of another in order to construct scenarios that consider both correlation strength and probability. In addition to IP similarities and matching ports, the frequency of alerts is assessed. In the end, an attack graph is generated from the extracted attacker strategies. ⋄*Tags: G{prediction, intelligence, correlation, detection}; T{network intrusion}; I{alerts}; D{similarity, ontology}; A{contextual}; K{yes: supervised learning, SVM and neural network classification, behavior graph extraction}; S{aware}; APT{1,3,6: network activity}.*

A more recent paper by AlEroud and Karabatis [7] (also see [125]) presents a layered attack detection system that utilizes semantics and context through a semantic network describing relationships between attacks. Their approach uses Conditional Entropy Theory – the uncertainty of one event given another [285] – to create attack context profiles that filter out non-relevant events. In addition, the Anderberg similarity coefficient measure [82] is used to detect similarities in binary network data. For example, the 'host context' profile considers type, vulnerabilities, operating system, applications, and services when filtering. ⋄*Tags: G{prediction, correlation, detection}; T{host intrusion, network intrusion}; I{network traffic}; D{ontology}; A{contextual}; K{yes: supervised learning, Bayesian classification, semantic network extraction}; S{aware}; APT{1,3-7: general approach}.*

Traffic Flow Analysis Solutions

Flow-based approaches are increasingly used to detect network attacks. Sperotto et al. [295] offer a good overview of traffic flow IDS systems. Over the years, a number of different solutions have been proposed:

Münz and Carle [232] present TOPAS, a traffic flow and packet analysis system compatible with Cisco NetFlow and IPFIX. TOPAS provides a framework for user-defined detection modules operating in real-time. The data used is netflow-specific in nature: Source and destination IP addresses/ports as well as protocols are considered. The system's detection algorithm encompasses threshold-based detection via pre-defined values that need to be exceeded, principal component classifiers (PCC) to detect anomalies in multivariate time series, outlier detection through the comparison of a sample to previously learned, normal behavior, and rule learning through a classification extracted

from these "good" and "evil" training sets. ⋄*Tags: G{detection, analysis}; T{network intrusion}; I{network traffic}; D{anomaly, pattern}; A{contextual}; K{yes: supervised learning, outlier classification}; S{aware}; APT{1,6: network flows}.*

Vance's work [322] is one of the few approaches that focus directly on APTs: He describes a flow-based monitoring system that uses statistical analysis of captured network traffic data to detect anomalies. He uses change detection through sketch-based measurement [165] to identify flows that hint at command & control traffic, data mining, or exfiltration activities. Volume, timing, and packet size are of primary interest; the respective baseline and subsequent analysis consider packet and traffic throughput, number of concurrent flows, TCP/IP SYN and RST packets, flow duration, and the current time. ⋄*Tags: G{intelligence, detection}; T{network intrusion}; I{network traffic}; D{anomaly}; A{contextual}; K{yes: change classification}; S{aware}; APT{6: specific network flows}.*

Another example of flow-based intrusion detection is described in [8]: The authors' approach utilizes probabilistic semantic link networks (SLN) synonymous to graphs using similarity values to describe node connections. The target's IP address, time and duration of communication, as well as various other features such as protocols and flags are mined from network traffic and the corresponding flows. Common characteristics are translated into link weight between the respective nodes of the graph. ⋄*Tags: G{prediction, intelligence, correlation, detection}; T{host and network intrusion}; I{system logs, app logs, network traffic, alerts}; D{similarity}; A{contextual}; K{yes: decision tree classification, semantic network extraction}; S{axiomatic}; APT{1,3,6: general approach}.*

Web Traffic Detection and Analysis Systems

Web service attacks are often part of the reconnaissance stages of a targeted attack or aim to publicly disclose sensitive information. More often than not, business-critical infrastructure can be accessed through exposed web portals that allow privileged users to monitor or configure backend systems. Because of their high visibility, web servers are also frequent targets of defacement attacks. For these reasons, a number of security solutions focus on the detection of malicious HTTP traffic:

Razzaq et al. [257] describe a system able to detect and classify web application attacks. Threats are specified through semantic rules that establish the context: Both attack consequences and common application properties such as protocol use are evaluated. The system analyzes the user part of an HTTP request (header, message) to detect authentication bypass attacks, DoS, probing, cookie stealing attacks, and more. The authors developed an ontological model (see also [256]) using a description logic based on OWL [209] and validated through OntoClean [121]. The inference rules were implemented using the Apache JENA framework [16]. ⋄*Tags: G{intelligence, correlation, detection, analysis}; T{network intrusion}; I{network traffic}; D{ontology};*

A{contextual}; K{yes: deductive learning}; S{axiomatic}; APT{1,3,6,7: specific network traffic}.

Earlier work of Razzaq et al. [255] describes another interesting semantic approach. The authors introduce an application-level IDS using a Bayesian filter to find malicious scripts in HTTP protocol traffic. URL, parameters, cookies, etc. are considered. This 'spam filter' assigns values to specific keywords and generates a score. Potential attacks are matched with attack descriptions stored in an external database. ⋄*Tags: G{detection}; T{network intrusion}; I{network traffic}; D{ontology}; A{contextual}; K{yes: deductive learning, Bayesian classification}; S{aware}; APT{3,6: specific network traffic}.*

Another semantic intrusion detection system is presented by Sangeetha and Vaidehi [76]. It defines malicious behavior as rules that consider source, frequency of occurrence, and parts of the HTTP packet content. Rules are represented as BNF grammar [96]. Fuzzy Cognitive Mapping [163] is used for attack prediction. The authors' traffic sniffer and interpreter system was implemented as part of a client-server infrastructure. ⋄*Tags: G{prediction, detection, analysis}; T{network intrusion}; I{network traffic}; D{pattern, similarity}; A{contextual}; K{no}; S{aware, denotational}; APT{3,6: specific network traffic}.*

SpuNge [22] is a system explicitly designed to detect targeted attacks through behavior clustering (similar behavior with respect to malicious resources used) and location/industry correlation. The analysis focus lies on URLs – its framework is able to detect machines that are part of the same attack. This is achieved through hierarchical clustering and string similarity measurement utilizing the Levenshtein distance [177]. SpuNge determines host distance (hostname similarity) and request distance (request path similarity) and groups the processed requests accordingly. ⋄*Tags: G{intelligence, correlation, detection}; T{host and network intrusion}; I{network traffic}; D{similarity}; A{contextual}; K{yes: hierarchical clustering}; S{aware}; APT{1,3,6: specific network traffic}.*

Thakar et al. [306] also focus on the analysis and extraction of traffic patterns. Unlike SpuNge, their work revolves around the extraction of signatures that can later be used by an IDS. Specifically, the authors log SOAP [326] traffic and extract information such as client identifiers, IP addresses, ports, and certain strings in HTTP requests and response packets. The data is then clustered using an SVM-based classifier [64]. Extraction and analysis utilizes the Longest Common Substring (LCS) algorithm [18]. ⋄*Tags: G{intelligence, analysis}; T{network intrusion}; I{network traffic}; D{pattern}; A{attribute}; K{yes: SVM-based classification}; S{aware}; APT{1,3,6: specific network traffic}.*

Zarras et al. [352] introduce BotHound, a detection method for malware communicating over HTTP. The system automatically generates models for benign and malicious requests and classifies new traffic in real-time. The primary goal is to discover bot traffic and C2 communication through suspicious header chains (sequences of HTTP headers)

and HTTP templates that encompass content data such as IP addresses, ports, transported file types, and more. Like in [22], string similarity is measured using the Levenshtein distance [177]. ⋄*Tags: G{detection}; T{malware}; I{network traffic}; D{pattern, similarity}; A{attribute, behavioral}; K{yes: supervised learning}; S{aware}; APT{6,7: specific network traffic}.*

Attack-specific Approaches

There are a number of network attack detection systems that focus on a specific type of threat. Because of their prominence, (distributed) *Denial of Service* (D/DoS) attacks are of special interest. One such solution is introduced by Gamer et al. [110]. Their proposed attack detection system is placed on routers and focuses on DDoS attacks and malware propagation. Network traffic is sampled and then refined in several stages: At the first level of granularity, Gamer's approach only considers the overall number of packets and attempts to find anomalous changes in volume. Level two differentiates DDoS from worm propagation by analyzing target subnets. Protocol anomalies hinting at a specific attack are detected in stage three. This e.g. includes the anomalous ratio between incoming and outgoing packets that typically accompanies a DDoS attack. ⋄*Tags: G{detection}; T{network intrusion}; I{network traffic}; D{anomaly}; A{behavioral}; K{no}; S{aware}; APT{3,7: specific network traffic}.*

'Vanguard' [197] is a detection system addressing low-rate and random-interval DoS attacks. Luo at al.'s approach is formal in nature: They propose a detection scheme for polymorphic DoS attacks that registers anomalies in TCP traffic. The decision is primarily based the ratio between incoming data and outgoing ACK packets. Vanguard has been implemented as Snort [56] preprocessor plug-in, presenting a more specific and better tested approach than Gamer's model [110]. However, its narrow focus on low-rate DoS attacks limits its use; the applicability of the algorithm to other types of DoS attacks has not yet been explored. ⋄*Tags: G{detection}; T{network intrusion}; I{network traffic}; D{anomaly}; A{behavioral}; K{no}; S{aware}; APT{3,7: specific network traffic}.*

On the ontology side, the system by Ansarinia et al. [15] provides an interesting take on the detection of DDoS attacks. The authors model prerequisites and consequences (e.g. unwanted disclosure) of such attacks and automatically generate an attack ontology based on Mitre's CAPEC [219], CWE [222], and CVE [221] threat information. Events and IDS logs are combined and converted to a single format: CEE [220]. In the end, it is possible to ontologically describe how a vulnerability is comprised of certain weaknesses and how exploiting them leads to a successful attack. ⋄*Tags: G{detection}; T{network intrusion}; I{threat info, alerts}; D{pattern, ontology}; A{contextual}; K{yes: deductive learning}; S{denotational, axiomatic}; APT{1,3,6: general approach}.*

2.5.3 Multi-Source Domain

Multi-source approaches typically focus on data fusion and event correlation. The primary categories within this domain are SIEM-like solutions and log correlation systems:

Logs are record files created by a wide range of devices and applications. Their primary purpose is non-repudiation through event auditing as well as system diagnostics. *Log analysis* systems retrieve log files and assess their contents. This can again be done using pattern or anomaly detection (see IDS) and will typically yield simplified alerts or a list of anomalous log entries. Their simple yet versatile nature makes bulk-processing easy and feasible for a wide range of applications.

Log analysis is often used in conjunction with multi-source event fusion. Solutions such as *Security Information and Event Management* (SIEM) systems take log files of e.g. several intrusion detection systems, traffic flow analyzers, and OS event logs to correlate or visualize attacks. SIEM systems usually do not monitor assets on their own; they merely process logs, alerts, and other monitoring reports generated and supplied by other tools. Multi-source (log) analysis is the closest thing to an attack interpretation system currently on the market.

SIEM systems and SIEM-like event fusion tools have become increasingly important in today's cyber-defense. SIEM development has spawned open source solutions like OSSIM [9] as well as various commercial products. Combined with the assessment of conventional log files, multi-source event aggregation and correlation is a promising new approach to understanding cyber-attacks.

SIEM-like Systems

SIEM-like systems are prototypical of the multi-source domain. One of the earliest multi-source approaches was introduced by Gorodetski et al. [116]. Their general paper on multi-agent system (MAS) technology for IDS encompasses attack simulation and intrusion detection learning. A model mapping attacker intentions to actions as well as targets is introduced and formally described by the authors. The proposed simulator considers network traffic data, data from the OS audit trail, system logs, and application audit data. Combined attacks with e.g. shared source IP addresses are detected through pattern matching on pre-processed input streams. Specific learning algorithms are mentioned but not explained in detail. ◇*Tags: G{intelligence, detection}; T{host and network intrusion}; I{system logs, app logs, network traffic, alerts}; D{pattern}; A{behavioral}; K{yes: supervised learning, Bayesian classification, visualization}; S{aware, denotational}; APT{1,3-7: general approach, attack simulation}.*

A general APT attack model following the intrusion kill chain [133] is presented by Bhatt and Gustavsson [33]: The logging module collects security events as well as various logs and submits them for analysis. Malware forensics is part of the framework but not specified in detail. A dedicated intelligence module is responsible for event correlation and searching. An experimental implementation was realized on a 5-node

Apache Hadoop cluster and tested with constructed log files. ⋄*Tags: G{intelligence, analysis}; T{malware, host and network intrusion}; I{system logs, app logs, alerts}; D{pattern}; A{behavioral}; K{no}; S{aware, denotational}; APT{3,5-7: general log analysis approach}.*

The system proposed in [205] aims to establish real-time situational awareness through semantic event fusion to detect multi-stage attacks. Event streams from intrusion detection systems are correlated with pre-defined alert templates and mapped to various categories (e.g. scan or intrusion), services (e.g. web, FTP), protocol stacks, and consequences (e.g. DoS). Attack criticality is also modelled. Unlike Bhatt's primarily time-based approach [33], Mathew et al.'s basis for correlation are similarities of IP addresses and semantically linked events. The authors implemented their system using the model editor FUME [204] and the fusion engine INFERD [300] on an emulated OSIS network [114]. ⋄*Tags: G{intelligence, correlation}; T{network intrusion}; I{system logs, app logs, alerts}; D{pattern, similarity}; A{contextual}; K{no}; S{aware}; APT{1,3,6: network activity}.*

Atighetchi et al. [21] present 'Gestalt', a cyber-information management system that simplifies the access to event data stored on various systems. Unlike classical SIEM solutions, Gestalt leaves the data where it was generated. The focus lies on forensics: the actual methods and techniques required to access the data are abstracted and described using a new Cyber Defense Language (CDL). All information is provided via a single interface accessible from a central management workstation – something that is achieved through the use of the Asio tool suite [26] and the web ontology language OWL [209]. Queries are submitted using SPARQL [327]. ⋄*Tags: G{intelligence, correlation, response}; T{host and network intrusion}; I{system logs, app logs, network traffic, event traces, alerts}; D{ontology}; A{contextual}; K{no}; S{denotational}; APT{1,3-7: multi-system data correlation}.*

Sadighian et al. [272] propose an alert fusion approach that incorporates public vulnerability data (CVE, NVD) and contextual attack information such as network configurations, host settings, application requirements, and user-specific configurations. This data is retrieved from a dedicated configuration management system. The collected information is then converted to a unified format and combined with common IDS alerts. A pre-populated set of ontologies for alerts, context information, and vulnerabilities is used as basis for the subsequent decision process. Fusion rules are built using SWRL, a language based on the OWL description logic (OWL-DL). In contrast to other solutions, Sadighian's system focuses on alerts collected by different sensors but describing the same event. This potentially reduces redundant and irrelevant information. ⋄*Tags: G{detection, analysis}; T{network intrusion}; I{threat info, network traffic, alerts}; D{ontology}; A{contextual}; K{no}; S{denotational}; APT{1,3,6: network activity event fusion rules}.*

Event Correlation Solutions

The second focus of multi-source data analysis is correlation. Unlike SIEM systems, the focus here lies more on the underlying correlation algorithms and less on data generation or management considerations. This promises high synergy potential between solutions of the two categories.

In [109], the authors describe the application of data mining techniques to identify patterns in data. A combination of association analysis, which aims to discover interesting relationships, and similarity-based propagation analysis is used to fuse log events into semantic incidents. Host data, user data, and events from both the OS and antivirus software are used as input for classification. ◇*Tags: G{intelligence, correlation}; T{malware, host intrusion}; I{system logs, app logs, event traces}; D{pattern, similarity}; A{contextual}; K{yes: deductive learning}; S{aware}; APT{4,5: host event discovery}.*

Langeder [175] introduces a framework for dynamic threat recognition and combines it with a proof-of-concept classification comparing Bayes, SVM, and decision trees. Patternized rules are extracted from a training environment and include attributes such as IP addresses, time, HTTP status and request information, ports, users, and more. Best results were achieved with the SVM approach; however, processing performance was only assessed for small data sets. Further domain testing is required to generically compare the various classification methods. ◇*Tags: G{intelligence, correlation}; T{network intrusion}; I{app logs}; D{pattern, anomaly}; A{contextual}; K{yes: unsupervised learning, SVM, decision tree, Bayesian classification}; S{aware}; APT{1,3,6: network event classification}.*

Bordering the domain of network-based correlation solutions, the work by Debar and Wespi [72] addresses IDS weaknesses such as event flooding, lack of context, false alerts and lack of scalability. The authors propose an intrusion detection architecture that correlates the output of several host-based and network-based probes to produce a condensed view of an incident. Next to the definition of conceptual and operational requirements, an alert class hierarchy considering both target and probes is presented: Generic information and probe status messages are combined with victim host information such as process or port details. ◇*Tags: G{intelligence, correlation, analysis}; T{host and network intrusion}; I{alerts}; D{-}; A{contextual}; K{no}; S{aware}; APT{3-7: host and network alert correlation}.*

Strasburg et al. [301] introduce S-MAIDS, a semantic model for automated tuning, correlation, and response selection in IDSs based on observable attack indicators the authors call 'signals'. Each signal is decomposed into a domain/characteristic such as the high-level protocol used (e.g. TCP), a type constraint (e.g. integer), and an value (e.g. 80). The proposed model is formalized using OWL. Cross-system correlation was assessed using IIS log messages and the output of a Netflow-aware system. As a reasoning-based ontology, S-MAIDS requires predefined attack responses to be present in the knowledge base. ◇*Tags: G{intelligence, correlation, response}; T{host and net-*

work intrusion}; I{app logs, network traffic}; D{-}; A{contextual}; K{no}; S{aware, denotational}; APT{3,6: app and network event correlation}.

2.5.4 Semantic Domain

Unlike the other three categories, this section focuses solely on formal definitions and general ontology models. Only approaches that specifically revolve around formal definitions, ontologies, and other semantic approaches are reviewed here. Also, this section includes articles that cannot be clearly attributed to one of the other domains. Semantics-aware solutions designed to support attack detection on the host or network can be found in the respective subsections above.

A number of models and some select data providers focus on semantics-based detection of cyber-attacks or malicious behavior in general. Most of the time, the term 'semantics' is used very loosely and only refers to the process of assigning meaning to e.g. specific patterns of system functions or network packets. Other solutions identify sequences of code that produce identical results.

While almost every solution discussed in this chapter can be considered semantics-aware, this particular section focuses on semantics-based approaches per our definition found in Section 2.2.2.

General Semantic Systems and Ontologies

Semantic systems and ontologies have found resonance with the information security community a good while ago. Landwehr et al. [174] were among the first to define a taxonomy of computer program security flaws. While that was not an ontology per se, their work laid the foundation for later attack models. Raskin and Nirenburg [253] eventually introduced a semantic approach to information security in regards to the unification of terms and nomenclature. Today, several application-independent semantic systems can be found in the literature:

Razzaq et al. [256] describe a general approach to ontology-based attack detection and argue its suitability for web application security purposes. While they also propose a domain-specific model (see [257] in the overview tables), the paper primarily introduces general ontology engineering methodologies such as 'Methontology' [98]. A layered model for ontology design is presented.

The paper by Anagnostopoulos et al. [11] is a workable example for the application of semantics to general intrusion scenarios. The authors seek to classify and predict attacker intentions using a Bayesian classifier and a probabilistic inference algorithm. Their semantic model includes both legitimate and illegitimate actors, the formal characterization – dubbed behavior – of the actor, activities in the form of sequential events, concrete commands issued, and an overall state of attack triggered by specific commands. ◇*Tags: G{prediction, intelligence}; T{host and network intru-*

sion}; I{-}; D{pattern, ontology}; A{contextual}; K{yes: deductive learning, Bayesian classification}; S{axiomatic, denotational}; APT{3-7: general approach}.

Yan et al. [348], on the other hand, focus on the conversion of raw sensor alerts into a machine-understandable format in order to enable easier data fusion. They advertise the use of a Principal-subordinate Consequence Tagging Case Grammar (PCTCG) that considers object, location, method, cause, 'has object' and 'is part of' rules, attack stage, and attack consequence of an intrusion. Together with a 2-atom semantic network [323], their system is able to generate attack scenario classes that can be extracted and used as detection templates for e.g. IDS systems. ⋄*Tags: G{intelligence}; T{host and network intrusion}; I{alerts}; D{ontology}; A{contextual}; K{yes: deductive learning, grammar-based classification, semantic network extraction}; S{axiomatic, denotational}; APT{3-7: general approach}.*

Languages and Models

There are a number of languages that form the basis for many a semantic model. Meier [211] offers a good overview of approaches by introducing a model of attack signatures for use on pattern detection systems. It is based on a meta-model for the semantics of database events by Zimmer and Unland [359]. The author identifies several types of information relevant for misuse detection: Exploit languages used to encode attacker actions, event languages that represent information to be analyzed by an IDS (e.g. HiPAC, SNOOP, NAOS, and ACOOD), detection languages used to describe signatures (rule-based (e.g. P-BEST), state-transition-based languages (e.g. STATL, IDIOT IDS), algebraic languages (e.g. LAMBDA, ADeLe, and Sutekh), response languages (alert information), and report languages such as the Intrusion Detection Message Exchange Format (IDMEF) [73].

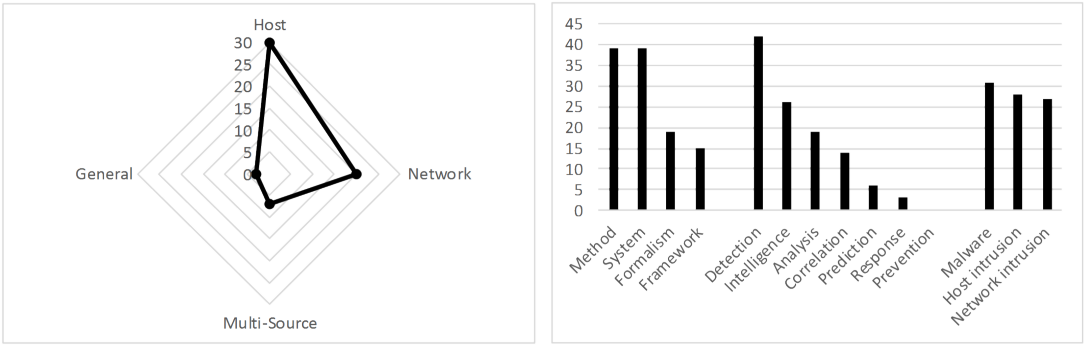
For example, Totel et al. [312] correlate events from different IDS sources to combat the usually high number of false positives. The authors developed ADeLe, a language specifically tailored to describe exploits and attacks from the target's perspective in addition to the intrusion response. Their correlation model supports event sequences, (non-)occurrence, recurring events, and time constraints. While ADeLe is not a data provider or analysis system, its event correlation capabilities make it especially useful for describing multi-stage attacks. ⋄*Tags: G{intelligence, response}; T{host and network intrusion}; I{-}; D{-}; A{-}; K{no}; S{denotational}; APT{3-7: attack description language}.*

2.6 Results

In this section we investigate the general and statistical findings derived from this survey of 60 domain articles. This includes scoring per the quality assessment introduced in Section 2.3.4, a list papers identified as especially good fit for APT detection efforts, as well as the full categorization (see Section 2.4) of all solutions.

Paper	Year	Paper	Year	Paper	Year	Paper	Year
[72]	2001	[52]	2007	[108]	2010	[7]	2013
[116]	2003	[232]	2007	[132]	2010	[22]	2013
[347]	2004	[356]	2008	[37]	2010	[272]	2013
[312]	2004	[54]	2008	[76]	2010	[301]	2013
[349]	2004	[238]	2008	[205]	2010	[257]	2014
[13]	2005	[275]	2008	[106]	2010	[213]	2014
[47]	2005	[243]	2008	[120]	2011	[346]	2014
[11]	2005	[287]	2008	[141]	2011	[322]	2014
[53]	2005	[50]	2009	[79]	2012	[8]	2014
[358]	2005	[1]	2009	[334]	2012	[175]	2014
[110]	2006	[255]	2009	[171]	2012	[33]	2014
[197]	2006	[109]	2009	[66]	2012	[21]	2014
[32]	2006	[25]	2009	[15]	2012	[128]	2014
[143]	2007	[263]	2009	[199]	2012	[352]	2014

Table 2.1: Publication date breakdown



(a) Domain focus (b) General categories

Figure 2.4: Overview of paper categorization and domain

2.6.1 Findings Summary

The mean date of publication of the surveyed papers is 2009. There are two deviations from the rule set by the publication date constraint (2003): the key papers [170] and [72] were included for their overall contribution to the field despite their more advanced age. See Table 2.1 for a breakdown of publication dates.

Interestingly, the focus on specific techniques changed little over the years. For example, system event traces are still widely used as basis for threat detection and analysis. Of the 30 papers published more recently (after 2010), 40% process system events. This stands in contrast to a 41.7% overall ratio. The same can be observed for a number of categories, including learning techniques. Even the use of semantics-based approaches did not significantly increase. Shy of 57% of the post-2010 papers can be classified as at least partially context-based; this is only slightly higher than the overall average of 53.3%. The trend towards semantic solutions is recognizable, but not notably so.

In contrast, 9 of the 13 top-scored (score > 7) papers that have been awarded the highest score in APT domain applicability (Q5) were published in recent years. This

Paper	Key research	Domain	Score
[141]	Host-side C2 traffic identification and behavior extraction through graph mining (Jackstraws)	Host	8.5
[263]	Heuristic classification of dynamically generated application traces (Malheur)	Host	8.5
[79]	Automatically created application profiles through graph-based function/parameter tracing	Host	7.0
[213]	Connection discovery for actors, machines, and malware through code semantics and function behavior (VirusBattle, BinJuice)	Host	7.0
[238]	Emulation of human reasoning process to distinguish user from attacker actions	Host	7.0
[257]	Detection and classification of web app attacks by means of an ontological model	Network	8.5
[352]	Detection of C2 traffic through malicious and benign HTTP traffic templates (BotHound)	Network	7.5
[15]	Attack ontology generation based on threat intelligence information and event fusion	Network	7.0
[22]	Behavior clustering through industry/location correlation based on URL strings (SpuNge)	Network	7.0
[322]	Flow-based traffic monitoring system capable of statistical anomaly detection	Network	7.0
[21]	Ontology-based data management system for central forensic data analysis (Gestalt)	Multi-source	7.5
[116]	Attacker intention modelling and traffic simulation system considering multi-source data	Multi-Source	7.0
[312]	Event correlation through attack/exploit language describing the target's view (ADeLe)	General	7.0

Table 2.2: Papers scoring highest for TA relevance (Q5 = 1.0; Overall > 7.0)

shows that the focus is slowly shifting towards the detection and analysis of targeted attacks, even though many of the articles were not authored specifically with APTs in mind. As prime examples for techniques that can more easily be transferred to this new class of threat, these 13 papers are of special interest to answering research question R4 thanks to their determined scores. See Table 2.2 for a complete list.

In addition to particularly APT-relevant articles, highly graded works are of special interest as well. In Table 2.3, we take a closer look at papers scored 8.0 or higher. Because of the constraints defined in Section 2.3, all of the research has been classified with a TA (Q5) score of at least 0.5.

Delving deeper into the individual stages of a targeted attack, we see that exploitation, installation, C2, and action stages are roughly equally represented. We see 34 to 37 methodologies that directly or indirectly contribute to phases 4-7. The reason can be found in the nature of common malware: A larger number of malicious software variants include functionality that is comparable to the operations implied by the APT kill chain seen in Figure 2.1. Initial (exploit) code execution, installation activity, remote communication and payload execution is not necessarily unique to targeted attacks and can be spotted by a wide range of tools that consider host or network activity.

The delivery phase (3) is in the scope of 50% of the surveyed papers. However, as delivery may include common e-mail communication and local device activity, most of the solutions only indirectly consider it – more focused work specifically targeting delivery actions is still rare.

Paper	Key research	Domain	Score
[53]	Static malware detection algorithm incorporating semantics and depicted as control-flow graphs	Host	8.5
[141]	Host-side C2 traffic identification and behavior extraction through graph mining (Jackstraws)	Host	8.5
[263]	Heuristic classification of dynamically generated application traces (Malheur)	Host	8.5
[54][106]	Malware classification and behavior extraction through dependence graphs (MiniMal, Holmes)	Host	8.0
[166]	Host-based function anomaly detection system with added context	Host	8.0
[346]	Data flow graphs depicting execution flow of various objects (DAVAST)	Host	8.0
[257]	Detection and classification of web application attacks through ontological model	Network	8.5

Table 2.3: Papers with the highest overall score (Overall $>$ 8.0)

Reconnaissance (phase 1) is covered by 15 solutions, most of which are capable of analyzing web traffic that could yield insight into suspicious scanning activity or targeted information mining from publicly accessible resources. Stage 2 (weaponization) is currently not in the scope at all, as it takes place solely on the attacker's systems. Here, alternative approaches such as active intelligence into code reuse or post-attack attribution need to be investigated.

Taking a closer look at the distinctive semantic categories defined in Section 2.2.2, we most often see semantics-aware solutions (44 papers). This is followed by solutions that include formal denotations of languages or rules (denotational semantics, 14 papers). Axiomatic and operational semantics are still rarely used (6 and 4 papers, respectively).

In Section 2.7, we discuss the implications of these findings and present a model of a workable APT detection system based on key components identified in the surveyed literature.

2.6.2 Statistics

Primary domain (see 'General categorization') is a unique property used for initial *categorization*. Each article can be assigned a single domain. Of the 60 papers surveyed, 30 present a host-based solution, 20 focus on the network, 7 describe multi-source approaches, and only 3 could not be classified or were independent of a specific application.

Contributions most often encompass methods or models and a (prototypical) tool or system (39 papers each). 19 papers introduce a formalism or language while 15 papers revolve around a more general framework. See Figure 2.4 for a breakdown chart and Table 2.4 for a full overview.

The goals of the respective solutions noticeably lean towards threat detection: 42 papers discuss approaches specifically aimed at detecting attacks or spotting attack indicators. Threat intelligence or classification components could be found in 26 articles. The actual analysis of a threat – done usually to determine its nature or goal – is an integral part of 19 papers, followed by 14 works that also consider threat correlation or

event fusion. Only 6 solutions attempt to predict threats, and not a single one focuses on preventive measures. Threat response is also rarely found: only three articles claim to support follow-up activities to a previously detected attack.

In contrast to the diverse goals of the surveyed approaches, the type of threats combated is almost evenly distributed between malware (31 papers), host intrusions (28 papers), and network intrusions (27 papers). This shows that countering each means of attack is considered equally important by the research community. See Table 2.4 for details.

Data collection is distinctively dominated by dynamic methods. 51 papers describe dynamic techniques to acquire input data. While data gathering capabilities do not necessarily mean that the solution records information by itself, a large number of solutions (47 papers) do in fact include monitoring functionality or at least a means to directly import results from integrated tools.

Static retrieval is used by 5 solutions, whereas 5 approaches do not collect data at all. 23 solutions dynamically monitor functions that are often logged as event traces. In contrast, 21 approaches sniff, record, or interpret network packets. Conventional logging is used in 8 cases. Code, disk, and memory monitoring are used least often – only three to four systems directly observe respective activities. In terms of type, capabilities mostly correspond to the approach identified above: We most often see system event traces (25 papers) and network traffic (23 papers). External alerts (11 papers), system and application logs (7 and 9 papers, respectively), and direct binary (raw) data input (11 papers) are used less often. General threat information is only collected in 2 cases.

Flow functionality is usually found in network-domain articles. Of the 47 papers that monitor information, 8 utilize data flow processing.

With the exception of emulation (7 papers), monitoring environments almost equally encompass native (17) and VM (14) systems. Interestingly, 14 papers do not specify a particular environment. The reason can be found in the nature of the works – many solutions are not yet part of an operational system but only roughly sketch a proposed functionality. Also, there exist a number platform-independent implementations. See Table 2.5 for details.

A total of 56 papers describe either *analysis* or *detection* functions – or both. Similar to data gathering, detection also leans towards dynamic techniques were the attack or sample under scrutiny is executed on an isolated system or is let loose in a testing network. More often than not, this analysis environment is also the one that monitors abovementioned data.

Pattern-based systems are most common (24 papers). Anomaly detection is used in 13 cases. This is only surpassed by ontologies – 15 solutions use various ontology models, languages, and tools to describe and detect a multitude of threats. Since this chapter expressively focuses on at least semantics-aware approaches, this number is not surprising. In accordance, distinctively semantics-based or context-aware solutions are

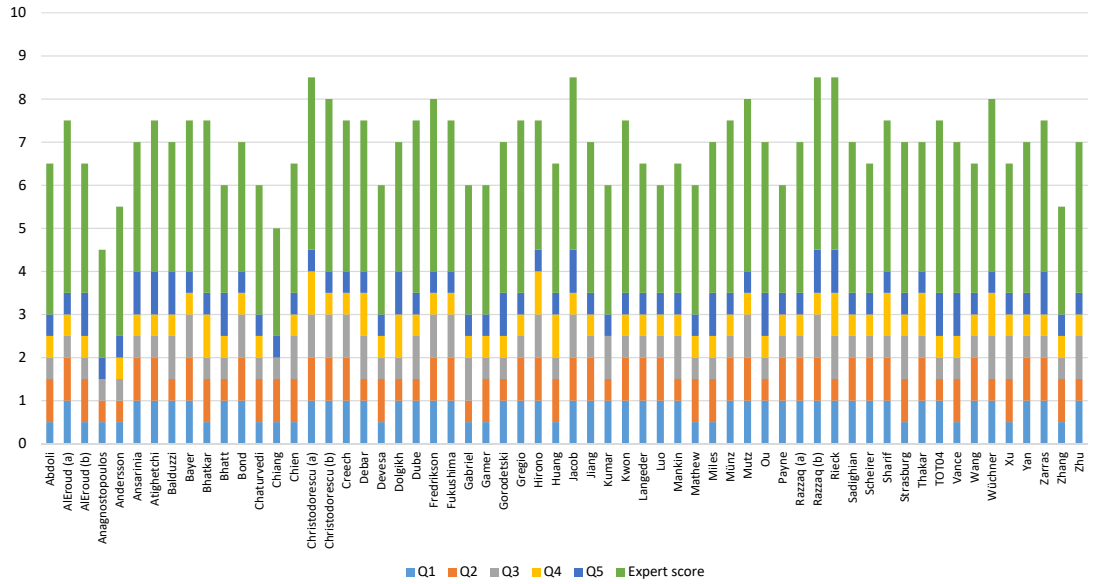


Figure 2.5: QA and overall scores for each surveyed paper.

described in over 50% of the papers. Less formal behavioral approaches (26 papers) are almost as common. At the same time the trend shows that attribute-based detection is on the decline (9 papers). This mirrors the opinion of many researchers that signature-based detection found for example in AV software is slowly becoming obsolete [230, 254, 198, 38].

On the processing side, on-demand analysis (28 papers) on either the local system (23 papers) or a central server (29 papers) is most widely used. Still, a total of 11 works claim to have achieved real-time processing. See Table 2.6 for details.

Knowledge generation is part of 32 of the surveyed solutions. Supervised learning is most widely used (14 papers), followed by deductive learning (7) and unsupervised approaches (2 papers). The specific methods used are diverse and do not lean towards one specific approach. Bayes-based systems are slightly more common than e.g. outlier or change detection (5 vs. 1 paper). On the extraction side, we most often see behavior graphs. Only five solutions offer post-analysis visualization. Part of the reason is the explicit exclusion of pure visualization systems. See Table 2.7 for a list of papers with knowledge generation capabilities.

2.6.3 Scores

Quality assessment was conducted through scores ranging from 0 to 10. The average score awarded to the reviewed papers is 6.94. Eight of the articles received an excellent score of 8 points or higher, while 38 papers were given a solid 7-point or above rating. Interestingly, the QA average (3.59 of 5) is slightly higher than the average expert rating (3.35).

In direct response to the QA questions, we have calculated average scores and assessed the number of papers that achieved the maximum score of 1 point for the

		[1]	[7]	[8]	[11]	[13]	[15]	[306]	[21]	[22]	[25]	[32]	[33]	[37]	[47]	[52]	[50]	[53]	[54]	[66]	[72]	[79]	[83]	[108]	[110]	[116]	[120]	[128]	[132]	[143]	[170]
GENERAL	Domain	Host	✓			✓					✓	✓		✓	✓		✓	✓	✓	✓		✓	✓	✓			✓		✓	✓	✓
		Network		✓	✓		✓	✓		✓						✓										✓	✓		✓		
		Multi-source							✓				✓								✓											
		General			✓																										
	Contribution	Framework				✓		✓	✓	✓			✓									✓					✓					
		Model/method	✓	✓	✓		✓					✓			✓	✓		✓	✓	✓			✓	✓		✓	✓			✓	✓
		Formalism		✓		✓	✓				✓	✓			✓			✓	✓	✓		✓									
		Tool/system	✓		✓		✓	✓	✓	✓	✓			✓		✓		✓					✓	✓	✓	✓	✓	✓		✓	
	Goal	Prediction		✓	✓																										
		Prevention																													
		Intelligence			✓	✓			✓	✓	✓		✓								✓	✓				✓					
		Correlation		✓	✓				✓	✓						✓						✓									
		Detection	✓	✓	✓		✓	✓		✓		✓			✓	✓		✓	✓	✓		✓	✓	✓		✓	✓	✓		✓	✓
		Analysis						✓			✓			✓			✓	✓			✓		✓	✓			✓	✓	✓		
		Response				✓			✓																						
	Threat type	Malware	✓								✓	✓	✓	✓	✓		✓	✓	✓	✓		✓	✓	✓			✓	✓	✓	✓	
		Host intrusion	✓	✓	✓	✓			✓	✓		✓	✓		✓						✓					✓				✓	
		Network intrusion	✓		✓	✓	✓	✓	✓	✓		✓	✓				✓					✓			✓			✓			

		[171]	[175]	[197]	[199]	[205]	[213]	[141]	[232]	[166]	[238]	[109]	[243]	[255]	[257]	[263]	[272]	[76]	[275]	[287]	[301]	[312]	[322]	[334]	[106]	[346]	[347]	[348]	[352]	[356]	[358]
GENERAL	Domain	Host	✓		✓									✓				✓		✓					✓	✓		✓			✓	
		Network		✓			✓																								✓	
		Multi-source		✓																		✓								✓		
		General				✓			✓			✓											✓						✓			
	Contribution	Framework	✓			✓			✓			✓	✓		✓	✓				✓	✓											
		Model/method	✓				✓	✓			✓	✓		✓	✓		✓	✓				✓		✓	✓	✓	✓	✓	✓	✓	✓	✓
		Formalism			✓				✓	✓	✓				✓		✓					✓	✓		✓			✓				
		Tool/system		✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓		✓	✓		✓				✓	✓	✓	✓	✓	✓	✓	✓
	Goal	Prediction										✓						✓														✓
		Prevention																														
		Intelligence		✓		✓	✓						✓		✓	✓						✓	✓		✓			✓			✓	✓
		Correlation		✓		✓							✓		✓		✓					✓									✓	✓
		Detection	✓					✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓				✓		✓	✓	✓		✓	✓	✓	✓
		Analysis	✓						✓				✓		✓			✓			✓			✓		✓						
		Response																														
	Threat type	Malware	✓		✓			✓	✓			✓	✓				✓			✓	✓				✓	✓	✓		✓	✓		
		Host intrusion		✓							✓	✓							✓			✓	✓			✓	✓			✓		
		Network intrusion		✓	✓	✓			✓			✓	✓	✓	✓		✓	✓	✓			✓	✓	✓					✓			✓

Table 2.4: General category by paper

		[1]	[7]	[8]	[11]	[13]	[15]	[21]	[22]	[25]	[32]	[33]	[37]	[47]	[52]	[53]	[54]	[66]	[72]	[79]	[83]	[108]	[110]	[116]	[120]	[128]	[132]	[143]	[170]
DATA COLLECTION	Data gathering																												
	Dynamic	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓		✓	✓		✓	✓	✓	✓	✓	✓	✓	
	Static															✓					✓								
	Approach																												
	Function	✓				✓				✓	✓			✓			✓	✓		✓		✓		✓	✓	✓	✓	✓	
	Packet	✓	✓	✓		✓		✓	✓														✓	✓	✓				
	Code												✓																
	Disk																										✓		
	Memory									✓							✓										✓		
	Logs			✓			✓					✓							✓					✓					
	Input type																												
	Threat info						✓																						
	System logs			✓				✓				✓												✓					
	App logs			✓				✓				✓												✓					
	Network traffic	✓	✓	✓		✓		✓	✓		✓			✓								✓	✓	✓	✓			
	Events traces					✓		✓		✓	✓						✓	✓		✓		✓		✓	✓	✓	✓		
	Binary/raw												✓			✓			✓		✓							✓	
	Alerts			✓			✓	✓				✓			✓									✓					
	Flow support			✓							✓			✓						✓									
	Available																												
	Environment																												
	Native	✓	✓						✓					✓		✓			✓				✓		✓	✓			
	VM					✓						✓	✓			✓									✓		✓	✓	
	Emulation									✓																			
	Unspecified			✓			✓				✓						✓	✓		✓		✓	✓	✓					

		[171]	[175]	[197]	[199]	[205]	[213]	[232]	[141]	[166]	[238]	[109]	[243]	[255]	[257]	[263]	[272]	[76]	[275]	[287]	[301]	[312]	[322]	[334]	[106]	[346]	[347]	[348]	[352]	[356]	[358]
DATA COLLECTION	Data gathering																														
	Dynamic	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Static						✓																								
	Approach																														
	Function	✓				✓	✓		✓	✓			✓		✓	✓	✓		✓	✓			✓		✓	✓	✓			✓		
	Packet			✓				✓		✓				✓	✓		✓		✓		✓			✓					✓	✓	
	Code												✓							✓							✓				
	Disk				✓								✓																		
	Memory												✓																		
	Logs				✓				✓								✓				✓										
	Input type																														
	Threat info											✓					✓														
	System logs					✓					✓	✓																			
	App logs		✓			✓					✓	✓									✓										
	Network traffic			✓				✓		✓				✓	✓		✓		✓		✓		✓						✓	✓	
	Event traces	✓					✓		✓	✓	✓	✓	✓			✓				✓				✓	✓	✓					
	Binary/raw				✓		✓			✓			✓							✓							✓			✓	
	Alerts			✓		✓				✓							✓											✓			✓
	Flow support							✓							✓	✓	✓	✓	✓		✓		✓				✓				
	Available																														
	Environment																														
	Native							✓						✓	✓	✓			✓				✓								
	VM				✓				✓				✓			✓				✓				✓					✓	
	Emulation								✓												✓				✓						
	Unspecified	✓		✓				✓																							

Table 2.5: Data gathering capabilities by paper

		[1]	[7]	[8]	[11]	[13]	[15]	[306]	[21]	[22]	[25]	[32]	[33]	[37]	[47]	[52]	[50]	[53]	[54]	[66]	[72]	[79]	[83]	[108]	[110]	[116]	[120]	[128]	[132]	[143]	[170]
ANALYSIS + DETECTION	Detection appr.	Dynamic	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	?		✓		✓	✓	✓	✓	✓	✓	✓	✓
		Static																	✓					✓								
	Detection method	Anomaly										✓		✓	✓					✓			✓	✓		✓						
		Pattern	✓		✓	?	✓	✓			✓		✓					✓							✓		✓	✓	✓			✓
		Similarity		✓						✓	✓																					
		Graph																	✓			✓										
		Ontology	✓	✓		✓		✓				✓				✓	✓												✓			
	Analysis technique	Attribute/property	✓					✓																	?			✓		✓		
		Behavioral				?					✓		✓	✓	✓		✓	✓	✓			✓	✓	✓		✓	✓		✓		✓	✓
		Contextual/semantic	✓	✓	✓		✓		✓	✓		✓	✓			✓	✓	✓		✓	✓								✓			
	Temporal domain	Real-time				?			?																?			✓			✓	
		Delayed						✓																								
		Interval								✓																						
		On demand	✓								✓	✓		✓	✓	✓		✓		✓			✓	✓			✓		✓			✓
		Unspecified		✓	✓	✓	✓						✓				✓									✓						
	Processing	Local		✓	?							?		✓	✓		✓	✓	✓	?			✓								✓
		Centralized		?	✓		✓	?	✓	✓	✓	✓				✓									?	✓	✓	✓	✓	✓		
		Distributed	✓										✓																			
ANALYSIS + DETECTION	Detection appr.	Dynamic	✓		✓	✓	✓	✓	✓	✓	✓				✓	✓	✓	✓	✓	✓				✓	✓	✓	✓	✓	✓	✓		✓
		Static					✓																								✓	
	Detection method	Anomaly		✓	✓				✓	✓														✓				✓				
		Pattern		✓			✓		✓			✓							✓	✓	✓									✓	✓		✓
		Similarity					✓	✓	✓			✓							✓											✓			
		Graph	✓					✓																	✓	✓						
		Ontology												✓	✓		✓				✓											✓
	Analysis technique	Attribute/property	✓				✓												✓	✓									✓			
		Behavioral			✓			✓			✓					✓				✓				✓	✓	✓			✓	✓		
		Contextual/semantic		✓		✓	✓	✓		✓	✓			✓	✓		✓	✓	✓		✓		✓	✓				✓				✓
	Temporal domain	Real-time			✓		✓		✓					✓	✓			✓						✓		✓			✓			
		Delayed				✓																									✓	
		Interval																	✓													
		On demand					✓	✓								✓	✓		✓	✓				✓	✓	✓	✓			✓	✓	
		Unspecified	✓							✓																		✓				
	Processing	Local	?	?			✓		✓		✓					✓			✓	✓					✓	✓	✓		✓		✓	
		Centralized			✓	✓		✓		✓				✓	✓		✓	✓					✓	✓						✓		?
		Distributed																														

Table 2.6: Detection and analysis capabilities by paper

		[7]	[8]	[11]	[15]	[306]	[22]	[25]	[32]	[37]	[47]	[54]	[66]	[79]	[83]	[116]	[171]	[175]	[232]	[166]	[255]	[257]	[263]	[287]	[322]	[334]	[346]	[348]	[352]	[358]	[109]	[141]	[106]
KNOWLEDGE GENERATION	Learning	Supervised	✓																															
		Unsupervised			✓	✓																												
		Deductive					✓						✓										✓	✓				✓				✓		
	Classification/ clustering	SVM					✓					✓						✓													✓			
		Decision tree		✓												✓		✓																
		Neural network											✓																		✓			
		Nearest prototype						✓																✓										
		Bayesian	✓		✓									✓			?		✓			✓											
		Markov							✓				✓	✓						✓														
		Grammars											✓	✓							✓							✓						
		Outlier																	✓															
		Change detection																							✓									
		Hierarchical clustering						✓															✓											
	Extraction	Dependency graph										✓														✓							✓	
		Behavior graph									✓			✓		✓								✓					✓		✓			
		Semantic network	✓	✓																								✓						
	Visualization	Available										✓		✓		✓								✓			✓							

Table 2.7: Knowledge generation through learning and classification

respective category (see Figure 2.5). It stands to notice that the attack domain (Q2) was usually well defined. 44 papers were given the full point; the average score was close to 0.87. In contrast, the definition and explanation of the type of operational data (Q4) used by the solutions was often found lacking. Only ten papers received the full score, resulting in an average grade of only 0.56. In regards to targeted attacks, 15 papers were found to be especially well-suited for APT defense-related objectives. See Figure 2.5 for a full score breakdown of all reviewed papers and Tables 2.2 and 2.3 for top picks.

2.7 Discussion

2.7.1 Research Questions

In this study, we have addressed 4 specific research questions that are now evaluated in this section. The overarching goal of the survey was to provide an overview of approaches and methods that could be employed to strengthen an organization's defense against advanced persistent threats and targeted attacks in general. Effective solutions tailored to the APT realm are very rare, presenting researchers with ample opportunity to develop specialized solutions that transport conventional cyber-defense mechanisms to this new threat domain.

In answer to R1 ("Which models, frameworks, formal definitions, and tools exist to describe information system attacks?"), we have identified various models, frameworks, formal definitions, and tools that describe information system attacks. The solutions and techniques they employ can serve as foundation for the design or technical implementation of a system capable of identifying targeted attacks at various levels: the host, the network, or a custom combination thereof. Armed with this knowledge, future research can be focused and prioritized in accordance to the user's specific needs in terms of data formats, specific approaches to gathering and monitoring as well as knowledge generation capabilities. A detailed evaluation and overview of all reviewed solutions can be found in Section 2.6.

In response to R2 ("Which semantics-aware and semantics-based tools and techniques exist to detect and evaluate attacks?"), we have highlighted solutions that use a semantic approach to perform their primary task. Semantics-aware tools are of significant interest when they establish attack context (see also contextual analysis technique in Table 2.6) and thereby enable research into the still problematic differentiation between common and targeted attacks. Semantics-based approaches offer additional insight and are often able to link actors and assets with a specific attack action. Enabled by the semantic categorization introduced in Section 2.2.2, we have highlighted solutions that focus on execution correctness/state transition, correlation through rules or I/O inference, as well as denotational approaches. This can help to e.g. better understand attacks and their consequences, identify high-level goals, or determine existing flaws in the defender's security design or implementation.

Techniques useful to analyzing targeted attacks are identified through their Q5 score as well as their APT stage affinity tag. This addresses research question R3 ("What are promising approaches to APT detection and how can they be classified?") by highlighting solutions deemed promising for APT/ATA detection and analysis. With semantics-based methodologies in the minority, there is still room for improvement in the areas of formalization as well as axiomatic and operational semantics. Next to awareness of meaning, knowledge generation was determined to be of utmost importance. Many papers offer functionality enabling the extraction of certain behavioral particularities that support the process of learning typical APT activity.

In accordance to R4, we determined which information is actually most helpful to the general task of detecting and understanding APTs by seeking to encompass all primary categories (see 2.4. In combination with the aforementioned APT stage tags, it becomes possible to maximize the scope through diversity of approach. With the aim to put together a system design checklist for a successful holistic defense strategy (see Table 2.8 for details), we infer that:

- The highest possible number of primary goals (G) should to be considered for maximum breadth of defense;
- Every threat type T needs to be countered as they are all part of a typical ATA;
- Each data input type I should be taken into consideration for a complete view on the system and its neighboring network infrastructure;
- Detection methods D and analysis techniques A need to be diverse to maximize obfuscation and evasion resilience;
- Knowledge generation K should be part of at least one solution per threat type so that the defender does not only detect, but also learn from adversary action on both host and network level;
- Each APT stage per Hutchins' model [133], possibly excluding weaponization, should be covered.

This can either be achieved through a combination of approaches – ultimately requiring correlation – or by utilizing a system general enough to be flexible in its application. The necessary information to assemble such a defense framework can be extracted from the reviewed and scored papers and is further detailed in the subsection below. We focus specifically on the kind of information required to identify targeted attacks and show how a system considering most of the APT stages as well as a diverse range of data collection and analysis methods could look like. To this end, we conceptualized the approach starting from the choice of data provider (data collection approach as summarized in Table 2.5) up to the correlation and interpretation of events. The resulting roadmap is intended to assist further research into applied APT defense and other domain-aware countermeasures.

2.7.2 APT Defense Framework

The approaches discussed in this study can be used to assemble a model of a system capable of dealing with a variety of targeted attack scenarios. While other defense mechanisms only return isolated events or counter single attacks, a holistic system would interpret the collected information and come to a more detailed verdict. Each satisfied item in our checklist would increase the confidence in the result.

The following roadmap details a suggested design process for a conceptual APT defense system based on the solutions reviewed in this survey. Please keep in mind that a concrete technical implementation (AIDIS) is discussed in later chapters, in particular Chapter 7.

We differentiate the following stages: Threat definition and modeling, formalization and ontology building, as well as data provider selection.

Threat Definition and Modeling

Prior to implementation, the defending organization will have to define both assets and threats. For threat definition, we found the model by Giuara and Wang [113] (see Figure 2.1) and the more common cyber kill chain by Hutchins et al. [133] to be simple yet effective solutions for modeling targeted attacks. The decision of which model to use largely depends on personal preference and data exchange requirements: While the cyber kill chain model considers command and control activity and weaponization as separate stages, Giuara's model is more detailed when it comes to the collection of data. Reconnaissance, exploitation, operation, and exfiltration stages are mostly identical, albeit named differently at times. Both models can be used in conjunction with MITRE's APT-enabled Structured Threat Information eXpression (STIX) data exchange format [223], which was developed to represent threat information in a comprehensive manner.

Before existing data sources can be combined and interpreted, the defender also needs to evaluate their own assets in order to determine likely targets. This organizational step goes hand in hand with an assessment of possible attacker goals and methods. The resulting top-down view on a potential targeted attack is the foundation for subsequent ontology building and goal mapping. For initial actor and asset evaluation, we propose the use of goal modeling techniques: KAOS [259], GRL [319], and the i* Strategic Dependency (SD) model [92] are promising candidates for implementation. Subsequent technical solutions will want to add reasoning to the mix – this is where ontologies come into play.

Formalization and Ontology Building

Ontologies were identified as a promising way of approaching the challenge of formalizing threats and threat responses in a semantics-aware manner. Evaluating the top results

shown in Tables 2.2 and 2.3, we can see that ontologies and related semantics-based methods are among the more highly scored solutions [213, 238, 256, 15, 21].

The information security community has only in recent years begun to truly embrace the concept. Originally a discipline of philosophy, ontologies in information science have become a formal approach to describing data types, properties, and interrelationships of entities within a specific domain. Their reasoning capabilities and data formats set them apart from semantics-unaware relational databases. Depending on general requirements and desired granularity, system designers can choose from numerous languages or systems. Data formats and languages include RDF, OWL, OWL-DL, and SWRL, among others. In the reviewed papers, ontology building and design often relies on established implementations like Protégé [296] or Apache JENA [16]. Queries to an ontological system are usually written in SPARQL [327].

For an APT detection system to succeed, research will have to bridge the gap between pure formalization and event interpretation by combining various data sources into a domain-complete attack ontology. Only then can we collect all available information and assign it a place in the greater picture. Fox et al. [104] emphasize that an ontology needs to be designed with a number of competency question in mind. Each question is representative for the information the completed ontology is supposed to answer. For designing an APT defense system, these questions would have to revolve around data providers and specific attack activities:

- Which data providers do I need to utilize in order to be able to detect a satisfyingly high number of targeted attack stages?
- Which data provider is able to detect which attack stage or activity?
- What are the techniques used in an activity and how do they translate to data provider events?
- Is the current activity indicative of a targeted attack, and if yes, which?
- Who are the actors of the attack and which assets are targeted/affected?

Fused into one system and complemented by a suitable reasoning engine such as the ones offered by various Protégé plugins [296], such an ontology would be able to answer straightforward, natural-language questions about any hypothetical or suspected ongoing attack. For this to work, it is necessary to fill the ontology with detailed knowledge of the attacks and assets identified in design stage 1. Only then can we move on to the selection and implementation of data providers.

Data Provider Selection

Once the APT ontology has been defined and supplied with the necessary knowledge as well as inference rules, we can begin to link each attack stage or activity to specific events. As argued by Hutchins et al. [133], countering the different stages of an attack will require both analysis and detection systems. The asset owner will have to choose a number of data providers capable of collecting both host-based as well as network-based

information. It is prudent to employ systems that, put together, encompass as many types of input types (see Table 2.5) as possible.

The reasons are manifold: Reconnaissance and delivery will likely involve networked resources and may not even register on the host. Since different layers of an organization's network may be accessed or analyzed by the attacker without him actually copying or manipulating resources, the traffic passing through the LAN as well as the implied general behavior is of particular interest. Flow-based approaches would help identify anomalous communication patterns while suspicious web traffic could be a sign of initial probing activities. Alerts that are harmless by themselves could be correlated to other findings using a dedicated SIEM-like system.

Sooner or later during exploitation, installation, or action stages, the attacker will interact with a local system. Here, host monitoring and malware detection comes into play. Again, the defender will not have to reinvent the wheel to protect himself from isolated malicious activity. The true challenge is registering events that do not appear to be harmful by themselves, but might be part of a larger attack. While data collection and correlation are well-researched, mapping individual events to a greater picture is still a challenge. This study shows that attack semantics is a term used in many a work, but that there is often a significant gap between promise and delivery.

Source data will also be one of the key success factors of any practical APT defense implementation – inadequate or too low a number of data providers will make it nigh impossible to identify attacks affecting specific or multiple targets. On the other hand, it might not be feasible to include too many sources lest the system would experience slowdowns or a decrease in interpretation accuracy. An effective implementation will have to carefully consider available solutions and hand-pick a small number of providers suited to the respective task. This survey identified monitoring and data classification [54, 263, 257, 22, 322, 346] as well as attack description, profiling and extraction to be a vital part of this stage of APT identification [79, 141, 312, 346]. Various detection systems spread across all the primary detection domains will help to assemble the picture [53, 352, 322].

To further support data provider selection and to offer a simplified view on the surveyed solutions, we introduce a design checklist based on the categorization used in this study (see Table 2.8). Each key property has been awarded a minimum coverage threshold (in square brackets) that should be satisfied by the chosen data providers.

Once data providers have been selected based on specific needs, the actual semantic engine can be built around the objective of detecting, explaining, and locating targeted attacks. Developing such a system is the main goal of this very thesis.

2.7.3 Limitations of the Literature Review

To limit the scope of the survey, the articles reviewed in this chapter have solely been chosen using the criteria defined in Section 2.3. Additional literature catering to specific

	Prediction	Prevention	Intelligence	Correlation	Detection	Analysis	Response
G met [5/7]							
	Malware	Host intrusion	Network intrusion		Attribute	Behavior	Context
T countered [3/3] K for T [3/3] Correlation for T [3/3]				A used [3/3] G for A [3/3] Corr.: A [3/3]			
	Threat info	System logs	App logs	Network traffic	Events traces	Binary/raw	Alerts
I incorporated [5/7]							
	Recon	Weaponization	Delivery	Exploitation	Installation	C2	Actions
APT stage covered [5/7]							

Table 2.8: System design checklist. G...goal, T...threat type, A...analysis technique, K...knowledge generation, I...input data type

topics not covered by the initial search terms was usually not considered, even if certain aspects were later identified as potentially relevant, such as solutions focusing primarily on knowledge generation.

Note that papers published after 2015 were not considered in this literature review due to the dissemination date of the journal article [187] corresponding to this chapter. Additional works are discussed in the respective 'Related Work' sections.

2.8 Summary

The detection of advanced targeted attacks is highly interwoven with more conventional approaches to intrusion or malware detection. In this chapter, we identified 60 articles that introduce methods, models, frameworks, formalisms, or applied systems that could potentially contribute to the defense against APTs and other multi-stage cyber-attacks.

With ontologies and general semantics-based approaches, an increasing number of attack models have found their way into the field of information security. The shift towards APT-aware solutions is noticeable, but not as pronounced as suggested by many a title. We identified only a total of 13 papers (see Table 2.2) that could contribute significantly to the fight against advanced attackers, while the remainder of research mostly provides tools aimed at more common or highly specific cyber-threats. Still, each of the reviewed articles has something unique to offer and should be considered when developing new systems.

To simplify prioritization, the categorization introduced in this chapter provides researchers with the means to conveniently browse for solutions best suited to mitigate or model particular APT scenarios. This is complemented by the design concept of a defense framework which uses input from various data sources to detect and analyze a targeted attack. We specifically used the resulting checklist to define the requirements and capabilities of our own APT detection system.

Chapter 3

Methodology

Contents

3.1 Research Design

The methodology chosen to investigate our approach to targeted attack detection and interpretation is *experimental*, *constructive* research based on large-scale data analytics as well as top-down threat modeling. The research design follows the Design Science Research (DSR) process by Vaishnavi and Kuechler [320], which consists of five phases: 1) Defining relevant real-world problems, 2) research into related work that attempts to address the issues, 3) development of specific solutions offering additional value over the current state of the art, 4) evaluation of the developed solution to determine usefulness and applicability, followed by a 5) concluding write-up.

Research in this thesis is founded on two cornerstones: Formal modeling and threat definition using a top-down threat–response approach, and a technical bottom-up component focusing on knowledge generation and anomaly analysis. The reason for this choice can be found in the thesis' problem statements: In particular, we argue that the interpretation (i.e. semantic annotation) of intrusion events can only be achieved when combining data-centric research with high-level modeling efforts. Figure 3.1 summarizes the overall approach.

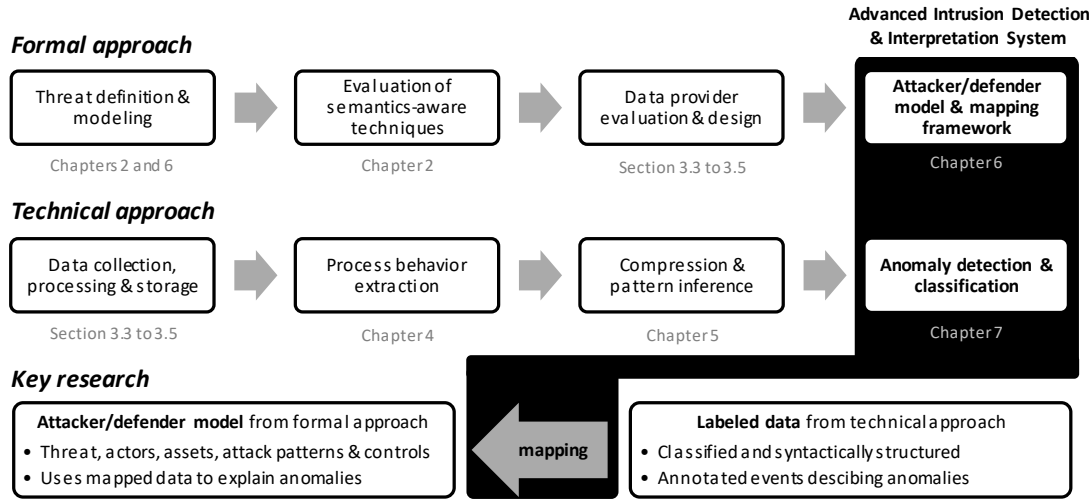


Figure 3.1: Stages of research. Both a formal and a technical approach was pursued in this research.

3.1.1 Formal Approach

After the initial definition of the research problems to be addressed by this work as part of DSR phase 1 (see Section 1.2), our formal approach can be divided into the following 4 stages:

1. **Top-down definition of APTs and their individual stages of attack.** In order to provide a template for future threat modeling, we aim to research and develop a universal definition of APT-class attacks by separating them into granular stages that are more likely to correspond to individual attack actions than common high-level models.
2. **Evaluation of ontological approaches and modeling languages.** Current ontologies and modeling languages are assessed to derive the requirements and design criteria for the development of an APT model able to consider all key aspects of a targeted attack. In addition, semantics-aware tools and methods are surveyed, determining their suitability for APT mitigation.
3. **Assessment and classification of data providers.** Once APT stages and modeling criteria have been established, data providers capable of capturing on-system activity corresponding to the compiled threat definitions are researched. The goal is to ascertain which classes of tools provide the necessary type and abstraction level of information for an effective intrusion detection solution supporting model-based event annotation.
4. **Research into a holistic attacker–defender model**. In this final stage of the formal approach, the actual APT attack–defense model is created, enabling analysts to enrich real-world attack data captured as part of the technical approach with meaningful threat semantics.

These formal aspects are designed to complement the experimental, data-driven approach introduced below.

3.1.2 Technical Approach

While the formal approach revolves around modeling the very threats we aim to mitigate, the technical side explores data monitoring and processing, behavioral inference and analysis, as well as anomaly detection and classification. Specifically, our bottom-up effort comprises four stages:

1. **Observation of actively used computer systems.** As part of the initial data monitoring process, use an agent corresponding to the previously identified data provider classes in order to collect data from machines connected to both a local network (LAN) and the Internet. This is followed by data processing and storage in preparation for later anomaly detection and classification.
2. **Extraction of behavioral patterns.** Subsequently, we extract behavioral information from the collected data to determine the most relevant processes and features in regards to anomaly detection. Ubiquitous processes and their interaction with the operating system are prioritized as part of our efforts to reduce the system's reliance on identified malware samples.
3. **Transparent and reversible data compression.** Our research plan includes ongoing efforts to reduce the computational requirements of expensive processing steps. To this end, we develop a mechanism that compresses reoccurring events and helps analysts focus on behavioral deviations.
4. **White-box anomaly detection and granular classification**. Finally, we use the collected dataset as well as domain knowledge gleaned from all previous formal and technical stages to create a system able to spot attacks and classify them in accordance to their attack stage or pattern. In combination, our research is designed to provide the means of mapping specific monitoring data to a model of threat actor behavior as part of a novel semantic enrichment process.

In the following, we discuss the design principles applied to each of the conducted (data-driven) experiments.

3.2 Experiment Design

This research is founded on a number of experiments that are designed to prove the efficacy of the AIDIS approach. In general, each component of our system is first evaluated as individual solution for e.g. anomaly detection or knowledge extraction. Only then is the system assessed in its entirety. In order to be as censorious as possible we follow a number of experiment design principles that ensure our results are representative:

- **Using a unified data set across all experiments.** For evaluation purposes, we ensure that the type of data processed stays the same across all experiments. Only the amount of information available varies throughout the project, as new traces are generated on a regular basis. Exceptions to this rule of uniformity are

allowed where they serve the validation of the process in question, e.g. when different datasets may prove a component's feasibility for alternative detection endeavors.

- **Separation of training and validation data.** The validation dataset will always differ from the traces used for training a component. We aim to use a 50/50 distribution ratio of training and validation data, as was found to be common practice for the systems reviewed in the literature survey.

In addition to the above principles, we decided to depict realistic, usually disadvantageous scenarios in regards to data selection. While this choice arguably reduces accuracy and/or performance in many an experiment, it also minimizes overfitting and helps represent data as it is likely to be generated in other real-world settings: None of the tests were tailored to anything more specific than the Windows platform. As a consequence, there is a lot of room for future optimization, since most results can be improved by simply relaxing some of the below design restrictions:

- **No data generalization beyond ID and user information.** In order to retain as much information as possible, we do not generalize directory paths, registry keys, or IP addresses. This decision was made to ensure reproducibility even if it negatively impacts performance. Possible generalizations for a future, speed-optimized implementation are discussed in Section 8.2.2.
- **No removal of faulty or misleading data.** At the expense of overall accuracy we decided to refrain from removing data representing benign or malicious processes erroneously exhibiting behavior hinting at the respective other side of the spectrum. This includes crashing malware failing to execute its full payload, benign processes altered by a user in an atypical fashion, or data captured from a faulty session.
- **Minimization of the data basis.** To evaluate AIDIS's feasibility, we opted to base the initial assessment of its core component on a single process candidate automatically determined as part of the technical approach discussed above. For this ubiquitous process, only the first 10 seconds of activity are analyzed. The reason for this choice is mostly practical: Firstly, the at-scale assessment of all system processes over their entire lifetime is unlikely to be completed in close to real-time for a large dataset. Secondly, kernel processes are not commonly affected by malware delay mechanisms, as they only trigger when the respective functionality (e.g. the opening of a network socket) is actually used. Therefore, we hypothesize that observing start-up behavior is sufficient in most cases. Thirdly, we argue that the later analysis of additional processes can only increase overall accuracy as part of a performance trade-off. If experiment results already prove to be satisfying for a minimal dataset, it would better affirm the system's usefulness than too generous a selection.

In the following we discuss the nature of the data as well as the procedures applied to selecting, collecting, and (pre-)processing it.

3.3 Data Selection and Collection

We evaluated a total of 10 data providers during the selection process [188]. Data providers are tools that monitor and evaluate information collected on a host or network device. As argued by Hutchins et al. [133], countering the different stages of an attack will require both analysis and detection systems. An asset owner is encouraged to choose a number of data providers capable of collecting both host-based as well as network-based anomalies and events. It is prudent to employ systems that, in addition to local events, encompass e.g. kernel network events, raw traffic, or traffic flows. As discussed in Chapter 2, the reasons for this are manifold: Hostile reconnaissance and delivery actions will likely involve networked resources and may not even register in the filesystem.

In our research, we opted to use *kernel monitoring data* of moderate abstraction, which represents a compromise between raw events or API calls and high-level alerts based on often problematic patterns. Our final data selection comprised abstracted process and file system activity as well as network functions invoked on the endpoint. The instrument used for collecting the data is an unreleased monitoring agent prototype dubbed "Sonar", developed by IKARUS Security Software and Sonar Cyber Intelligence in cooperation with the thesis author's project team. It runs as part of the Windows kernel and utilizes SSDT hooking [120, 278] to remain undetected. Sonar collects process, thread, image load, file, registry, and network events in the form of abstracted API and system calls. These events describe i.a. process launches or terminations, access to local resources, or the retrieval of system configuration information. Other functions represent the establishing of network connections or general interaction with external resources — all of which can be considered an event in the context of a process. For more information about events, see Section 7.4.1 below.

The collected information is pushed to a central SQL database server every few seconds, where it can be queried and converted to both process trees and traces. Please note that the implementation of the agent and its database is not part of this thesis.

3.4 Data Processing

To maintain chronology as well as context, each individual event captured by the Sonar agent is time-stamped and can be linked to a specific process or thread through its respective ID. This allows us to construct trees of individual processes as well as entire system sessions, as discussed at length in [201].

Summed up, a process tree is a hierarchical representation of an executable and all the activity it displays during its lifetime. We use process events as root type since the Windows kernel provides all necessary information to associate an event to a triggering process through its process ID (PID). This works no matter which parent is responsible; child processes spawned via the command prompt can be linked together as can events

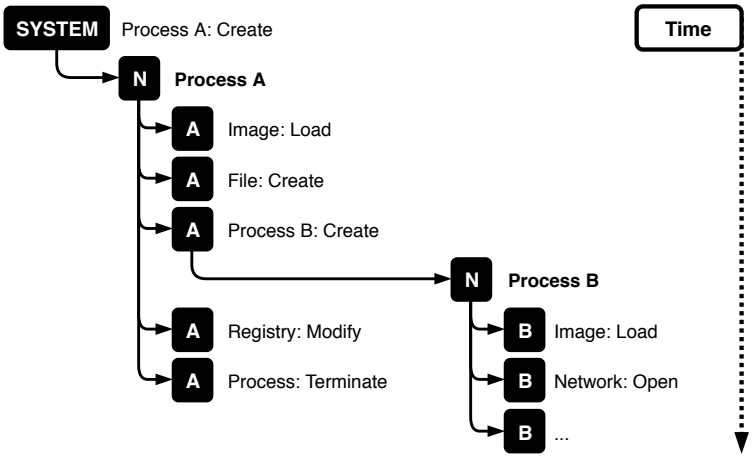


Figure 3.2: Example tree of a process launched by the kernel. Processing concurrency can cause child processes to trigger events at the same time as their parent, which makes it important to always maintain process context through (P)PID mapping to avoid intermixing.

induced directly from the kernel (i.e. by the `SYSTEM` process). Using PIDs in addition to timestamps to construct the tree is paramount to maintain process context. Using temporal information alone would result in events linked to wrong processes simply because of processing concurrency/multi-tasking. See Figure 3.2 for an example process tree.

The structure of the initially collated database of events can be described as multiple lists containing all recorded activity triggered by an arbitrary process. Each list (stored as table) has a column containing a timestamp that serves as primary sort condition. Since process events are the root type, tree building starts with the process event table. Every process event comes with the necessary PID, parent PID (PPID), and timestamp used to create or expand the tree. If the respective parent process is still running when the monitoring agent captures a snapshot, all information about that parent is readily available. However, because of PID reuse in Windows [49], new processes may be assigned a PID that has previously belonged to another, now terminated application. Building a process tree by using only PIDs can therefore corrupt the entire structure if the snapshot taken by the agent coincides with an instance of PID reuse. To avoid such collisions, start and termination times logged by Sonar for each process are also considered when constructing the tree, serving as a de-facto consistency check.

To further reduce the risk of incorrect parent–child assignment, the monitoring agent is typically started at system boot. This results in database entries that are *session-complete*, which makes it straightforward to find the correct parent process by simply comparing process launch times. This feature is especially vital when the analyst wants to assess the entire system state at a given time.

Ultimately, all process trees are converted to traces in order to enable word-based processing. The central elements for building process traces are again the IDs and timestamps logged for each event. First, all trees are saved as text in chronologically

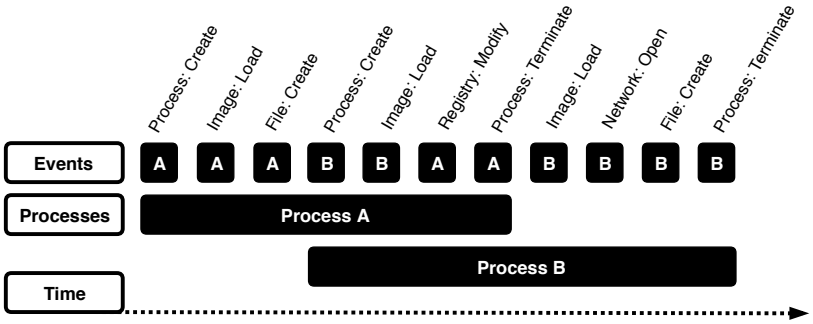


Figure 3.3: Orphan events in a context-unaware trace [190]. Smart traces counter the issue of intermixing events that belong to different processes or threads by rearranging them into a chronology by context (here: A followed by B).

ordered lists. If a tree contains two or more process events denoting the existence of children, a new list is created. Simply parsing it level by level would mess up the logical order and cause "orphan" events linked to the wrong process simply because of their timestamp, as depicted in Figure 3.3. To avoid this issue, we create a per-process timeline that starts with the initial process event followed by all activity associated to its PID in chronological order. A 'process terminate' event typically completes the list, provided the respective application has been cleanly shut down and is no longer running. Table 3.1 shows an example based on the tree depicted in Figure 3.2:

ProcessEvent	Start process A with PID 220
ImageLoadEvent	Load image "process.exe"
RegistryEvent	Write file "document.txt"
ProcessEvent	Start process B with PID 224
RegistryEvent	Modify registry key "HKLM/..."
ProcessEvent	Terminate process with PID 220
ProcessEvent	Start process B with PID 224
ImageLoadEvent	Load image "library.dll"
NetworkEvent	Connect to IP address 1.2.3.4
...	...

Table 3.1: Timeline of events as chronological trace ordered by process occurrence.

All AIDIS components use such "smart" traces for their pattern extraction and anomaly detection routines. With the potentially high number of events contained in a full session dataset, both trees and traces can grow to significant sizes. Despite the function abstraction implemented by SONAR, manually interpreting the depicted activity becomes a daunting task. Therefore, automated solutions are necessary to process the newly generated timelines. These solutions, combined into AIDIS, are central to this thesis and are discussed in depth in the following chapters.

3.5 Data Storage and Dissemination

We have created a full set of system monitoring data by observing 13 physical Windows 7 and Windows 10 machines actively used by software developers and office personnel. Our deployed kernel monitoring agent, which was developed in concert with the aforementioned company partner, logs various event types (process, thread, image load, file, registry, and network) to a central server that houses our evaluation data. Dedicated scripts realized in PL/SQL query the database and construct trees – or rather a tree-like representation of an executable including all the activity it displayed during its lifetime – as well as process-context-aware traces (sequential event lists following a per-process chronology) that are the basis for all further processing.

All data underwent company-side pseudonymization to protect the users of the monitored machines. Account names, host IDs, and IP addresses were replaced by hashed values; the added salt is protected by a passphrase known only to the partner company's works council. The corresponding keys to reversing the process were provided only when required by the evaluation process. All researchers working with the information signed an NDA stating that the data may not be passed on to a third party. Specifically, the thesis author agreed to not disclose any confidential information (documents, source code, software, documentation, customer data, etc.) to third parties, except to those who are required to access the information in order to evaluate or engage in discussions concerning a business relationship with the company. The thesis author agreed to refrain from reverse engineering, disassembling or decompiling any prototypes, software, or other tangible objects. This primarily applies to the Sonar monitoring agent used as main measuring instrument (data provider).

While the original data was stored on a system at the partner's premises, a local pseudonymized copy was provided for this research. Data querying and subsequent processing was done on virtual machines (VMs) hosted by St. Pölten University of Applied Sciences. User restrictions and hard drive encryption are in place to safeguard the information. The main VM, a dedicated data mining machine with 4 virtual cores à 2,400 MHz and a total 192 GiB of memory, is only accessible locally or through VPN. Some smaller, pre-existing datasets were executed on local university machines with fewer resources – these are identified in the respective 'Evaluation' chapters together with the specific samples used.

All findings will be or have been disseminated to the scientific community as peer-reviewed papers [187, 188, 189, 190, 191, 193, 194, 196, 201]. Raw and processed data that does not violate the aforementioned NDA is published alongside our findings. Prototypes will be released as open source code once the project officially concludes in March 2020. There are several follow-up projects in the pipeline that seek to continue research into aspects of AIDIS. Commercial exploitation is planned by Austrian IT security provider CyberTrap Software. Refer to the Conclusion (Chapter 8) for more information about post-project dissemination.

3.6 Summary

This thesis marries a formal attacker–defender model with applied anomaly detection and classification through experimental research. For analysis, we use abstracted Windows kernel monitoring data captured directly on the host. The key measuring instrument (data provider for most subsequent tasks) used in the research is a lightweight agent developed and deployed by our company partner. Based upon its output, we construct process trees linked by their timestamp as well as their process or thread IDs. The trees are ultimately converted to sequences (traces) of events mapped to the triggering process for context. All data, which was harvested from active company workstations over the course of six months, was anonymized and protected against unauthorized access in order to safeguard possibly sensitive user information.

Each of the following four chapters details one key component of AIDIS, the threat detection and interpretation system designed to answer the question of how advanced targeted attacks can be accurately identified using behavioral techniques.

Chapter 4

Sentiment Extraction from Structured Kernel Event Data

Contents

4.1 Introduction

Gathering insight about ongoing attacks through the detection and analysis of suspicious interaction of executable binaries with both operating system and network is key to successful endpoint defense. While most solutions focus on pre-identified samples of potential malware [330], we implement anomaly recognition based on ubiquitous OS processes. To maximize the efficiency of such an approach, we first need to identify programs with a higher tendency towards exhibiting malicious behavior within the context of a Windows user session. At the same time, a straightforward initial verdict about an app's activity can help prioritize future investigations and provides analysts with the means to quickly blacklist processes that are deemed malicious.

To achieve preliminary anomaly detection and relevant process identification, we propose a behavior-based classification solution that utilizes dynamic kernel event analy-

sis to learn and extract information from both clean and compromised Windows systems. Unlike many other machine learning approaches, our classification system is not a black box in terms of analysis and decision-making: We monitor and log kernel events that are stored and preprocessed in a central database. These events can be understood as abstracted system calls that describe the access, creation, modification, deletion/termination of specific processes, threads, files, libraries, registry keys, and network sockets. From this data we construct traces that consider the general order of events without fragmenting activity sequences that belong to the same process. The trace files are then split into bigrams of events. We use a supervised learning approach to determine the log likelihood ratio (LLR) of each bigram – this enables us to assign each event pair the likelihood of it occurring in sequence, as well as determine whether it is rather seen in infected or uncompromised software environments. Based on this ratio, a WordNet-like sentiment dictionary of events can be compiled.

We use a highly performant pattern matching approach to compare the extracted semantic signatures to unknown trace files, calculating an alignment score in the process. Using this score we are able to accurately decide whether or not an ongoing system session is affected by malware and pave the way for an even more detailed behavioral classification. Ultimately, it becomes possible to extract processes and event types that contribute most to the overall maliciousness of an observed system session.

The remainder of the chapter is structured as follows: In the 'Related Work' section, we take a look at similar approaches to malware detection and analysis. In Section 4.3.1, we describe the structure of the data generated by our kernel monitoring program and how it can be used to build n-grams out of smart process traces. Section 4.3.4 focuses on sentiment mining and the compilation of the dictionary and also discusses scoring. Section 4.4 takes a closer look at the implementation of the system and evaluates our initial results. Limitations and future work are discussed in the concluding sections.

4.2 Related Work

There are several scientific works that revolve around API or system call sequences as basis for malware detection and intrusion. One of the seminal works by Forrest et al. [102] describes a method for the detection of anomalies in Unix parameter-less system calls by building a database of common sequences. Somayaji and Forrest [293] build upon this approach to design an intrusion response system. The system of Mirza et al. [217] uses function hooking to monitor API calls. In their current implementation, they manually flag certain calls as suspicious and export the gathered information to a SIEM system. Miles et al. [213] focus on the interrelationship among malware instances to discover new connections between actors: Code and API call execution traces are compared to identify similarities. This includes websites, e-mail messages, and PE file headers. Xu et al. [347] introduce a waypoint mechanism in the form of markers on the execution path. This provides trustworthy control flow information for

subsequent anomaly monitoring. Waypoints are generated through static analysis and later imported as traps into the kernel.

Function call analysis is often used in conjecture with graph-based approaches. Kwon and Lee [171] introduce BinGraph, a system designed to discover metamorphic malware. It is postulated by the authors that the same API sequences occur in metamorphic variants of the same malware strain. Graphs matching is employed to search identical structures in the generated traces. Another graph-based approach, DAVAST, is described by Wüchner et al. [346]: The authors visualize trace information as quantitative data flow graphs where files, sockets, processes, and more are represented as nodes while the edges depict the execution flow. Rules define normal behavior as well as malicious activities.

In data analysis, calls or abstracted (generalized) events are often found in clustering and classification applications. Trinius et al. [314] and Rieck et al. [263] describe such an approach for malware behavior traces generated by dedicated dynamic analysis tools. The focus lies on API and system calls as well as their arguments. The used sandbox reports are embedded in a vector space to enable similarity assessment based on geometry. Hierarchical clustering is then used to identify groups of malware displaying similar behavior. Rieck et al.'s work later became 'Malheur', a system capable of rapidly processing (but not extracting) function call n-grams.

Another malware classification approach is proposed by Christodorescu et al. [54]: The authors came up with a method to mine behavioral data from trace files of benign and malicious samples. Both are collected during dynamic analysis and linked through their parameters. Christodorescu et al.'s system constructs dependence graphs for later comparison. Alazab et al. [6] combine n-gram analysis of API calls with a purely static approach. Unpacked malware is disassembled and API calls related to certain tasks are extracted. The most frequent function n-grams of known benign and malicious samples are used as feature set for later training and classification.

Sentiment analysis, as proposed in this chapter, subsumes a technique pertaining to text categorization, where the criterion of classification is the attitude (emotion) expressed in the text, rather than the "content" or topic [111]. Sentiment is usually extracted from natural language texts such as comments, feedback or critiques. Categorization either yields a simple positive/negative score or is expressed as an n-point scale [250]. As demonstrated by Pang et al. [240] determining the sentiment of an expression is not as straightforward as perhaps expected. The same holds true for a language of abstracted function calls: Each API or system call alone can be used for both good and ill – depending on its contextual and chronological vicinity to other functions.

Applying a sentiment approach to OS call traces is a novel concept. However, approaches similar to the LLR-based methodology presented in this chapter have been explored in other areas: Gamon [111] demonstrates automatic sentiment classification for customer feedback data. The author does not utilize semantic resources but rather

relies on SVM-based machine learning using a number of pre-tagged texts. Another approach by Wiebe et al. [341] replaces certain fixed-word n-grams with semantically stronger tags. Their goal is to identify subjective language and classification of instances in context by building upon methods like the one proposed by Hatzivassiloglou and McKeown [124], which determines whether two adjoining adjectives share the same alignment.

4.3 LLR-based Sentiment Extraction of Kernel Events

The extraction of sentiment, i.e. the tendency of a specific corpus of events towards varying degrees of maliciousness, provides an initial good/bad classification that helps identify suspicious processes, which are the basis for AIDIS' core anomaly detection and interpretation. In short, the approach discussed in this chapter uses supervised learning to create a dictionary of events that can be used to score unknown OS behavior. By extracting event pairs with the highest sentiment rating we can easily spot processes likely related to attacker activity.

4.3.1 Data Collection

As discussed in Section 3.4, the sentiment extraction process works with *event traces* collected directly on the hosts (endpoints). Event traces are typically defined as descriptions of operating system kernel behavior invoked by applications and, by extension, a legitimate or illegitimate user. More often than not, these events are abstractions of raw system or API calls that yield information about the general behavior of a sample [330]. API calls may include wrapper functions (e.g. `CreateProcess`) that offer a simple interface to the application programmer, or native functions (e.g. `NtCreateProcess`) that represent the underlying OS or kernel support tools. In the context of our system, process and network event data is collected directly from the Windows kernel. We employ a driver-based monitoring agent designed to collect and forward a number of events to a database server. This gives us unimpeded and fast access to events depicting various OS operations [201]:

- **Process events** – Whenever a process is started or stopped, the monitoring system registers a new event. Next to PID and paths, we record parent and contextual information such as ownership data. Process events are at the heart of our system as every other type of event is ultimately associated to a process.
- **Thread events** – Some events are triggered by individual threads instead of processes. The information logged by the agent is largely similar to process events; the main identifier for threads is the thread ID (TID).
- **Image load events** – Most processes load additional resources (functions) stored in various program libraries (DLLs). The nature of a DLL can give a good indication as to which behavior the executable will exhibit during its lifetime.

- **File events** – File events are logged when a file is read, created, accessed, modified, or deleted. Logging file interaction is important since processes can interact with virtually every file stored on the disk. Attack-related file events can e.g. help identify dropped executables or data theft.
- **Registry events** – Applications use the registry to save user and program settings while other hives contain startup programs or file type settings. Since it is a common target for espionage and system manipulation attacks, monitoring registry events is critical for any Windows-based detection solution.
- **Network events** – Network events encompass the handling of inbound and outbound connections as well as the access to general OS networking resources. Depending on the nature of the process, network events can be used as indicator for malicious behavior as many malware variants contact a remote system for e.g. command & control purposes.

The relative ease of monitoring as well as the semantic expressiveness of kernel events and network operations make these traces ideal for dynamic software and, by extension, malware analysis as well as application classification. The system introduced in this chapter uses this rich repository of behavioral data to compile sentiment dictionaries as well as graph-like star structures of event sequences that can describe not only a single application, but a system session in its entirety. This approach is detailed in the following subsections.

4.3.2 Preprocessing

All previously collected events are linked through their parent process in order to establish a semantic connection between action and cause. This is realized through two attributes that are present in all the data collected by the host monitoring agent: Creation time, and the PID that forms a unique identifier for each process. Threads work in a similar fashion. Like PIDs, thread IDs (TIDs) are appended to other event types (e.g. registry events). Both can therefore be used to construct an event tree depicting the flow of file system activity that helps to determine specific dependencies between processes or general events. Concatenated into a full system graph, the sequence of events of the monitored session can be assembled without orphan entries (depicted in Figure 3.3) interrupting the process flow. These so-called *smart traces* [190, 201] are the basis for all follow-up computation.

Further preprocessing includes the normalization of non-uniform IDs such as user names, security identifiers, and temporary folder names. This is done to make data more comparable across systems and to prepare the traces for anonymization.

4.3.3 Extracting Event n-Grams

N-grams are typically slices of words [45] with a length of n characters. A window with the size of n is slid over the string, creating a list of n sequential entries. If

applied to longer texts instead, one entry may take the shape of a word or even a sentence. In general, the number of corresponding n-grams is a measurement for the similarity between two strings. A Boolean feature vector for every string (where every dimension stands for an occurring n-gram) can also be used to determine the similarity by calculating the distance between two vectors. Even though n-grams are a fairly simple way to match texts, they are widely used for text analysis purposes because of their versatility and accurate representation of n-sized lists.

In this chapter we apply the n-gram approach to kernel event traces. Each event is part of a longer sequence – be that the trace of an entire session or a single process. The order of entries depicts dependencies or chains of events. We decided to use *bigrams* – kernel event pairs that are $n = 2$ in length – as the basis for later likelihood calculations. This decision was primarily based on accuracy and performance considerations: Unigrams ($n = 1$), while slightly more accurate for heuristic clustering purposes [201], do not retain any information about their immediate neighborhood, effectively reducing the desired dictionary to isolated terms. On the other hand, using n-grams with a size of $n > 2$ comes with the inherent risk of skewing the results without generating more expressive sequences: A slight variation in word order would result in differing n-grams unsuitable for describing a single event within the process context. Figure 3.3 exemplifies the issue: A context-unaware trace would result in n-grams that string together events which are likely unrelated or that randomly deviate in their place in time. Smart traces largely circumvent this problem by reordering all events into subtraces that are connected to their successors by a single transition event.

Since we are especially interested in extracting kernel events that are chronologically adjacent, using bigrams was the sensible choice. As added benefit, calculating the log likelihood ratio for n-grams of size $n = 2$ does not require consolidation, making it the most effective approach to tackle LLR-based challenges [84].

4.3.4 Sentiment Analysis

We use an approach akin to sentiment analysis [111] for generating initial knowledge about relevant OS processes. We use this optional stage to determine the most expressive process candidates for later investigation. At the same time, this first stage computes a first benign/malicious score that provides us with a tendency towards general harmfulness for the provided dataset. This verdict can also be used as additional feature in the final classification stage of the overall process, which is detailed in Chapter 7. In the following, we highlight the main properties of the sentiment analysis component, which has been initially disseminated in [190].

The likelihood ratio (LR) test employed here is a statistical method used to test model assumptions, namely the quality of fit of a reference (null) and an alternative model, whereby the simpler model can be understood as a special case of the more complex one [84]. The test assumes that the model is known, but that its parameters are not. The goal of a model is to find parameter values that in turn maximize the value

	A	**!A**
B	k_{11}=$k(AB)$	k_{12}=$k(!AB)$
!B	k_{21}=$k(A!B)$	k_{22}=$k(!A!B)$

Table 4.1: Event occurrence matrix K. The respective counts change in accordance with the number of times an event token is observed within the bigram.

of the likelihood function, which is equivalent to finding the set of parameters that make the data most likely [316]. The LR test also relies less on the assumption that a certain variable (i.e. word) is distributed normally throughout a text than e.g. chi-squared or z-score tests. When computing the occurrence of rarely observed events – which are at the core of many a malicious trace – likelihood ratio tests show significantly better results than the mentioned alternatives [84].

The *likelihood function* describes the probability that a given experimental outcome $k_1, ..., k_n$ is described by a model defined through n parameters p [228, 84]:

$$L(p_1, ..., p_n; k_1, ..., k_n)$$

Simplified into a single parameter, the likelihood function can be abbreviated as $H(\omega; k)$, where ω is a point in the parameter space Ω and k is a point in the space of observations K.

Dunning [84] defines the *likelihood ratio* for a hypothesis as the ratio of the maximum value of the likelihood function over the subspace represented by the hypothesis to the maximum value of the likelihood function over the entire parameter space, i.e.

$$LR = \frac{\max_{\omega \epsilon \Omega_0} L(\omega; k)}{\max_{\omega \epsilon \Omega} L(\omega; k)}$$

where Ω is the entire parameter space and Ω_0 is the hypothesis being tested.

As mentioned above, the basis for sentiment analysis are 'smart' kernel event traces reordered to maintain process and thread context (see Section 4.3.2). After tokenization, we extract bigrams (n-grams of length $n = 2$) from the collected sequences. The general goal of this stage is to find useful features in a large amount of system traces, in particular events that occur in combination. Inspired by Dunning's work [85], we compute the LLR score for each individual n-gram to highlight collocations characteristic for sequences of malicious and benign system events. This enables the analyst to ultimately rank the words of a corpus by their domain relevance and subsequently extract processes that are more likely to be involved in adversarial behavior.

The formulas presented in Table 4.1 (matrix K) are responsible for counting token occurrences in the given distribution. The different values are defined as the number of times both event tokens occur together (k_{11}), the number of times each event token has been observed independently from the other (k_{12} and k_{21}, depending on their position in the bigram), and the number of times the tokens were not present at all (k_{22}).

Once these counts have been determined, it becomes possible to compute the log likelihood ratio (LLR) score. LLR is generally easier to work with than the normal likelihood ratio, as results with values closer to zero describe a better fit [316]. Applied to the assembled matrix K, LLR becomes

$$LLR(K) = 2S(K) \cdot \left(H(K) - H\begin{pmatrix} k_{11} + k_{12} \\ k_{21} + k_{22} \end{pmatrix} - H\begin{pmatrix} k_{11} + k_{21} \\ k_{12} + k_{22} \end{pmatrix} \right)$$

where $S(K)$ represents the sum of all matrix elements k_{ij} in K and H denotes the function computing the Shannon entropy [285]. I.e. given an $m \times n$ matrix $X = (x_{ij})$, we have:

$$H(X) = \sum_{i=1}^{m} \sum_{j=1}^{n} \left(\frac{x_{ij}}{S(X)} \cdot \log \left(\frac{x_{ij}}{S(X)} \right) \right)$$

with

$$S(X) = \sum_{i=1}^{m} \sum_{j=1}^{n} x_{ij}$$

Using this methodology we are able to extract a list of bigrams that represent likely collocations. For example, the two image load events `kernel32.dll` and `kernelbase.dll` are commonly seen together in our dataset ($\sqrt{LLR} > 200$), signifying their semantic relationship. Other pairs describe sequences of certain processes or connect registry queries. Based on these calculations we can then determine the association of bigrams to different corpora of known benign and malicious events.

Dictionary Compilation

With a significantly high number of known benign and malicious kernel event traces at one's disposal it becomes possible to use the LLR scores for sentiment extraction and, in the process, for the compilation of a dictionary of suspect and valid events. To satisfy the requirements of reference and alternative model, we define the malicious data as special case of the larger, benign data set. This decision is based on observations that, for both clean and infected systems, the (unique) number of benign events will be greater than the number of functions used in a malicious context.

The dictionary compilation process itself is similar to the one described above. We determine the occurrence of known malicious n-grams in a benign sample set. In addition to LLR-based scoring performed for opposing corpora, we calculate the occurrence of partial and full malicious bigrams in the benign corpus. This results in a new LLR score that describes the association of each bigram to both sets of data. Mapped onto Table 4.1, we get the number of times an event token in the malicious corpus has also occurred in the benign set (k_{11}), has occurred partially in the form of the left or right half of the respective n-gram (k_{12} and k_{21}), and the number of times the malicious n-gram does not occur in the benign set at all (k_{22}).

Based on these scores we can assign sentiment to each identified bigram determined through the occurrence of the sequence and the maximum LLR measured for each corpus. The result is a normalized sentiment rating s ranging from $+1.0$ (benign) to -1.0 (malicious), whereas experimentation helped set the threshold for distinctively benign or malicious events to $s_t = -0.05$ for a reduced false positive rate. This approach yielded two distinct dictionaries with characteristic event pairs.

While a key aspect of our evaluation scenario, classification is not limited to good/bad decisions. It is similarly possible to compile dictionaries that describe certain malware families, semantic classes of attacks or even distinct benign software activities. The versatile nature of the process promises a number of future applications.

Sentiment Scoring

Scoring is achieved by means of both the benign and malicious scores determined through a comparison with the sentiment dictionary compiled in the previous stage. Unlike natural language implementations such as SentiWordNet [94], the malicious bigrams were not weighted from n-point to a flat score. Instead, the full range of values was utilized.

The final score s is determined by whether the root-LLR scores of the unknown event sequence lean toward the harmless or suspicious end of the spectrum. The calculation takes the sum of all root-LLR values observed in the dataset and divides it by the overall maximum root-LLR value in order to derive a percentage. This is done for both benign and malicious scores; in the end, the b-score is subtracted from the m-score for a final s.

Since there are no inherently malicious or benign functions in an operating system, most applications will exhibit behavior found in both domains. Still, LLR-scored sequence combinations have proven to be good indicators for both individual programs as well as entire system sessions. Simultaneously, the resulting scores give us a good idea which ubiquitous kernel processes are the most likely candidates for closer examination with one of the more advanced AIDIS components. Concrete results of our prototype implementation are discussed in Section 4.5.

4.4 Implementation

The utilized kernel monitoring agent (see Section 3.3 and 4.3.1) logs all the event types to a central listener that in turn writes the events to a Postgres database server. SQL is used to query the database and to construct the star structures that are the basis for all further processing. Our approach is able to selectively retrieve entire system sessions or pick out individual processes.

The smart traces used by the sentiment component are preprocessed and converted into matrices using Bash and Python scripts. LLR-based sentiment analysis is implemented in R [251].

4.5 Evaluation

There are two parts to this component's evaluation. First, we assessed the standalone *anomaly detection* capabilities of the approach, testing the dictionary creation and scoring system with both smart traces and conventional raw API call sequences used by many other solutions. We show that the first method is by far superior in accuracy and that the LLR component is well-suited for initial good/bad classification of event corpora.

In the second part (Section 4.5.2), the extraction of relevant processes from a large set of unique applications is demonstrated. We use the system to create a shortlist of ubiquitous Windows processes that contribute most to a benign/malicious decision, arguing that it is both reasonable and viable to base future analysis stages on these instances alone.

4.5.1 Anomaly Detection

The current prototype of the system was implemented in a test-bed environment consisting of 13 physical Windows 7 and Windows 10 computers used on and off by developers and office personnel of a medium business. For anomaly detection, this data spanned several weeks; process extraction conducted near the end of the project extended the available set 6 months of monitoring data (evaluation in Section 4.5.2). The partner company, a local security solutions developer, performed regular checks to ensure that the machines in question were not affected by undesired software. Additionally to the benign baseline (null model) dataset consisting of over 80 real-life system sessions including over 500 processes each, an isolated machine was manually infected with numerous malware samples of, but not limited to the Agobot, Beast, Blaster, Code Red, Conficker, Koobface, Loveletter, MyDoom, Sasser, ZeroAccess, and Zeus families and was studied over the course of 5 days. The variants used in the training stages were not utilized in the evaluation. Ultimately, the traces generated by that lab computer provided the alternative model dataset for our initial evaluation.

A total of 65.7 million events were recorded over that extended period of time, 700,000 of which were observed in a malicious context. Smart trace files were assembled as part of the pre-processing phase. After optimization, this process took about 1 hour on an Intel Core i5 4690S machine equipped with 8GB of RAM.

All further steps were conducted on an Intel i5-3210M machine utilizing 16GB of RAM. In that configuration, bigram extraction for the test data set was almost instantaneous. Aforementioned event files were reduced to 1.94 million benign and

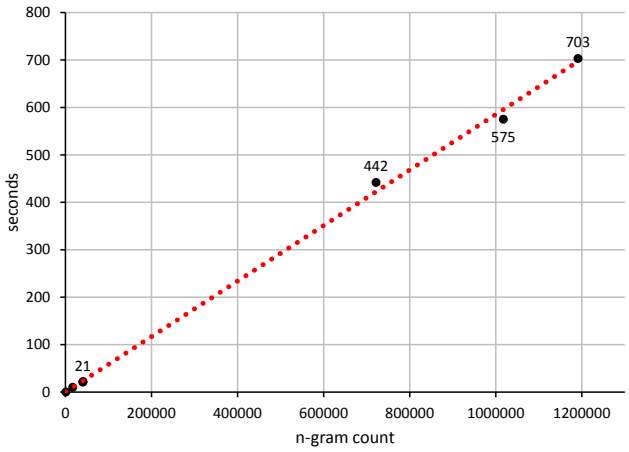


Figure 4.1: Performance analysis. The LLR process scales linearly in time.

13,000 malicious bigrams. The most resource-heavy stage – the calculation of all LLR values necessary for the extraction of sentiment – took a reasonable 10 seconds for the malicious corpus. The dictionary creation itself completed in 54 minutes. Please note that the more complex dictionary compilation only needs to be performed once per training set, while LLR processing is mandatory for each tested sample.

To demonstrate the universal applicability of the methodology as well as to compare performance of our smart approach versus a conventional trace, we additionally collected a large trace of raw system and API calls by monitoring the execution of benign and malicious software on a virtual Windows XP SP3 test machine running API Monitor[1]. Pre-processing removed all information other than the call name and its arguments while also replacing volatile memory addresses with dummy values. A total of 5,000 known benign and 800 harmful applications were analyzed in 10-minute intervals and combined to two non-smart input traces of 31 million and 45 million entries, respectively. Bigram generation reduced them to around 1 million unique event pairs each. For this larger dataset, LLR calculation could be completed in a mean 640 seconds, whereas the one-time compilation of the sentiment dictionary took 42 minutes.

Performance evaluation through regression analysis shows that LLR calculation scales linearly at $t = 5.872 * 10^{-4}n$, where t is the processing time and n is the number of bigrams extracted from a sample corpus. Considering these advantageous results, we expect to hit the memory ceiling sooner than any bigram corpus size limit imposed by computational complexity. See Figure 4.1 for a simple regression analysis.

Results

Smart event traces–During initial testing we extracted and scored likely n-gram sequences and compiled a sentiment dictionary containing over 1.94 million word pairs. Against this template we matched 83 new malicious sessions, of which 67 were correctly classified with a high confidence score of $s > -0.1$. Another 13 samples were awarded

[1] http://www.rohitab.com/apimonitor

Sample set	Trace type	Total	True	False	Undecided	TP/TN	Accuracy
Malicious events	Smart	83	80	0	3	96.4%	98.2%
Malicious API calls	Conventional	295	36	103	156	49.8%	73.0%
Benign events	Smart	81	81	0	0	100.0%	98.2%
Benign API calls	Conventional	282	274	8	0	97.2%	73.0%

Table 4.2: Scoring results for smart event and raw function traces. Evaluating smart traces is significantly more accurate than using unsorted API calls.

a low confidence score, while only 3 were scored as "undecided". On the benign side, 57 of the 81 harmless sessions were correctly classified with low confidence, while 24 were assigned a high-confidence score. None of the bigrams were incorrectly classified.

We performed a binary response ROC analysis extended by a confidence rating for the entire set of traces and determined an overall 98.2% accuracy even when all undecided cases are counted as low-confidence false positives. If they are discarded instead, the accuracy rating is pushed to 100%. This makes the sentiment system noticeably more precise than an earlier approach utilizing Malheur-based classifier training performed on unigrams [201].

See Table 4.2 for a score breakdown into the respective sensitivity and specificity values. Figure 4.2 shows the charted empirical ROC curve. Outliers with values far into the respective benign or malicious range are especially useful for the identification of behavior patterns, while close-to-neutral bigrams are indicative of activity inherent to both harmful and harmless applications. See Figure 4.3 for a plot of the specific results of both malicious and benign smart traces.

Raw function traces–Utilizing our alternative function call data set we compiled another sentiment dictionary containing approximately 2,2 million bigrams. It quickly becomes apparent that the approach is significantly less accurate than our new methodology: Of the 282 new benign function traces, 272 were correctly classified with a high confidence score, and 2 with a low confidence score. 8 samples were incorrectly classified as low-confidence malicious.

295 malicious function traces were matched against the dictionary as well. Of these samples, 27 were correctly assigned a high-confidence score, while 9 were scored with a lower confidence of $s >= -0.075$ & $s < -0.05$. 156 samples were determined as alignment-neutral and a total of 103 traces were incorrectly classified as benign. Combined into a ROC analysis, the overall accuracy for function calls is 73.0%, considering that 45 of the undecided cases lean towards an incorrect classification (low confidence) whereas the remainder is counted as tendentiously correct. Table 4.2 breaks down the scores for event and call type traces.

Discussion

It is apparent that conventional traces based on non-abstracted API and system calls are distinctively less accurate for sentiment analysis than our adapted approach. This

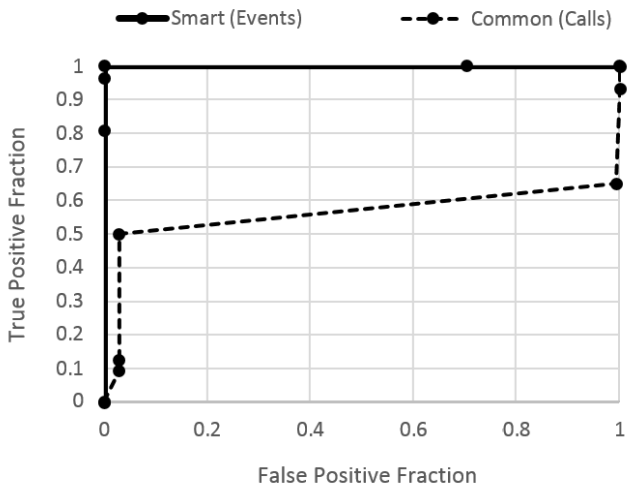


Figure 4.2: Empirical ROC curve for both trace types

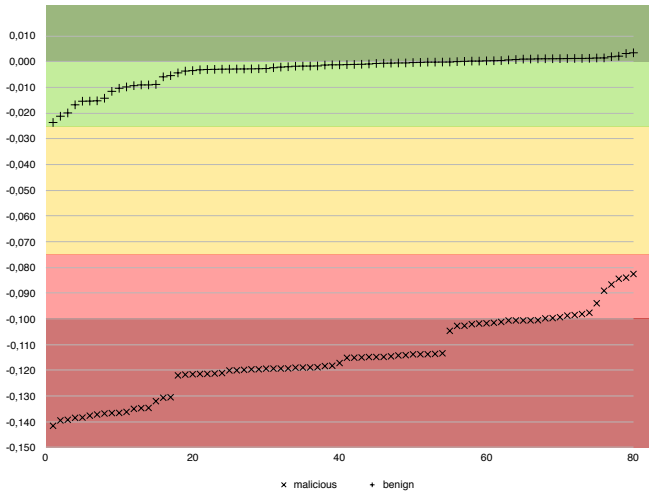


Figure 4.3: Smart trace scores mapped to decision thresholds

is due to the fact that, unlike smart traces and despite removal of memory addresses, raw system calls contain a lot of parameter information that is unique to each execution or a particular malware family. Bigram generation therefore results in a high number of pairs that are not overly representative for malicious software in general. Also, raw call data is far more vulnerable to mimicry attacks than the abstracted and combined behavioral data we utilize in our primary approach. This stems from the nature of the abstracted data: While it is easy to interject system calls into a strictly chronological sequence, the same is not likely to be successful in the case of smartly reordered events that describe actions rather than mirror them fully.

Thanks also to the fact that the introduced LLR methodology does not overcast malicious characteristics with the more numerous benign activities smart event sequences offer a far more accurate alternative. Above results show that abstracted event traces recorded over a longer period of time have a distinct advantage in both accuracy and performance over short-term malware analysis recorded on a function call level.

4.5.2 Relevant Process Identification

The identification of relevant processes happens in the optional 'sentiment analysis' stage of the AIDIS system (see Figure 7.2 in Chapter 7). The goal was to determine which known Windows OS processes are more likely to be part of a bigram with a high maliciousness score. If omitted, the selection of processes to be considered during anomaly detection and classification has to be conducted manually. In the following, the experiment and its results are discussed.

Results

The data investigated contained close to 2,000 unique processes (`svchost.exe` being one of them) that were launched during the lifetime of more than 10,000 benign and close to 2,000 malicious system sessions. In our benign testbed environment alone, 1,260 unique processes were observed across all workstations. Upon infection with APT malware, the test machine interacted with a total of 876 distinctive processes. In both cases, processes with identical names but deviating directory locations were counted individually, as malware often reuses common application names to evade detection.

The reason for the discrepancy between total events and the lower process counts lies in the nature of our data: Each process can trigger an arbitrary number of events during its lifetime, which ranges from minutes to weeks, depending on the frequency of system reboots. This is markedly pronounced for kernel processes that run for as long as the OS is active. Ultimately, we used only a fraction of the available data for anomaly detection to demonstrate AIDIS' feasibility in a single-process scenario. To get there, such a process had to be identified:

As a first step, all processes that occurred in only the benign or malicious domains were discarded, leaving a total of 120 executed binaries that are possibly ubiquitous. The reason for this is that with AIDIS, we want to reduce the reliance on any prior knowledge about the name or location of malicious software, focusing our anomaly detection efforts on omnipresent processes that will exist no matter the current state of compromise. Additionally, we argue that not all APT attacks will utilize dropped binaries that can be seen in the operating system; especially when campaigns involve manual intrusion activity or techniques such as process injection. Full paths were ignored during relevant process identification in order to make fake system services comparable to their genuine counterparts.

In step 2, we used LLR sentiment scoring [190] to determine the likelihood of a process occurring as part of an event bigram exhibiting malicious tendencies. With the threshold set to $s_t = -0.05$ and a tolerance bandwidth of 0.1, we extracted all processes with a mean sentiment rating of $s \leqslant 0.05$. This resulted in a reduced list of 84 processes. Next, we eliminated likely false positives that stem from the technical differences of physical and virtual machines, such as graphics drivers or first-run wizards that were initialized every time the virtual environment was started. With 72 processes left, it

Process (.exe)	Description	$\wedge$	$\overline{x}$	$\widetilde{x}$	$\vee$	$\sigma_{\overline{x}}$	k Events
conhost	Console window host process	-0.895	0.016	0.008	0.607	0.104	3,691.0
csrss	Win32 user-mode subsystem	-1.000	0.039	0.044	0.836	0.171	1,000.7
explorer	Explorer shell and file manager	-1.000	0.003	0.003	1.000	0.008	29,227.0
searchindexer	Windows search and indexing	-1.000	0.002	0.007	1.000	0.047	4,346.3
smss	Session manager subsystem	-1.000	-0.236	-0.332	1.000	0.416	6.3
svchost	Generic host process	-1.000	0.005	0.005	1.000	0.012	126,120.1
taskhost	Generic host process for libs	-1.000	0.022	0.011	0.983	0.048	1,871.7

Table 4.3: Shortlist of relevant processes identified through sentiment analysis. While `smss.exe` produced the most malign numbers, the amount of process data for a meaningful follow-up analysis was simply too small. We opted for the generic host process as something of a 'semantic worst case' with the highest amount of data (number of events in the database) available.

became apparent that user apps like word processors and programming environments need to be considered separately. While there is malware in the wild that utilizes such programs or their associated file types as injection targets or carrier medium, these attack scenarios were deemed too specific for the initial evaluation. Any baseline created for these processes would be highly dependent on user interaction which, in this case, might deviate significantly from benign instance to benign instance. At the same time, we hypothesize that even a compromised user application will eventually utilize a kernel process to perform a portion of its adversarial task – something that was corroborated by our tests.

With 57 processes remaining, we matched the OS-centric shortlist against two lists[1] [2] of ubiquitous kernel processes found in modern versions of Windows. This yielded a match for 7 processes that were determined to be relevant by the sentiment analysis component. In order to perform meaningful data mining, the process with the most recorded events (126.1 million) was chosen for all subsequent analyses: `svchost.exe`, the Windows generic host (service host) process [271].

See Table 4.3 for details about the 7 primary investigation candidates. Our results highlight that any event-based anomaly detection system might benefit from focusing on one or several of these processes.

Discussion

The multi-purpose `svchost.exe` process is involved in many operating system tasks and has existed since Windows 2000 [271], making it ideal for in-depth observation. At the same time, it arguably represents a worst case scenario for anomaly detection systems, as it is very difficult to create a behavioral baseline for such a versatile application. We argue that it is unlikely to find a Windows process that will produce more questioning results in an evaluation scenario, underlying the efficacy of AIDIS in adverse situations. At the same time, it needs to be emphasized that future iterations of the system will

[1] https://social.technet.microsoft.com/wiki/contents/articles/4485.windows-7-default-system-processes.aspx

[2] https://www.andreafortuna.org/dfir/forensics/standard-windows-processes-a-brief-reference/

include additional processes for reasons of diversification and accuracy, starting with the remaining 6 processes determined to be relevant by LLR. Additional applications such as user programs will be considered in corresponding scenarios. For example, malware delivery (deception) detection as per the APT kill chain is more likely to focus on processes like `winword.exe` and `acrord32.exe` while many installation, persistence, and launch tasks can be captured by observing `cmd.exe`, `regedit.exe`, or `net.exe`. Our initial evaluation focused on the generic host process because of its versatility but also due to data availability and time constraints. Refer to Section 7.5.3 and 8.1.1 for a closer look on computational performance.

4.6 Discussion

In the case of smart event traces it has proven difficult to find the optimal snippet size for scoring individual sample executions. We opted to use session traces in their entirety in order to facilitate anomaly detection and long-time activity analysis. Applying the approach to single event chains risks skewing the result of a binary good/bad decision, as the deviation in execution time between sentiment dictionary and analyzed sample is simply too great.

Despite the fact that the LLR algorithm regards rare occurrences as more relevant than tests that rely on a normal distribution of words [84], it is still possible that significant but rare malicious events are overshadowed by a mass of benign activity. For event traces, this issue can be countered by defining a time window for each investigation or by limiting dictionary compilation and scoring to a particular system process or process tree. In a raw system call context it is recommended to constrain the overall analysis time and create individual traces per investigated application. Special care has to be taken when regarding malware traces of unknown samples: If the sample did not execute correctly, we will likely not see any suspicious behavior other than the OS's routines for handling faulty applications.

Further testing with larger, more diverse data sets is essential to fine-tune the scoring process and to generate a truly representative dictionary for a particular problem domain. It stands to emphasize that our take on sentiment analysis is not necessarily confined to two-way decisions: Tiered scoring, classification by type or family, and other, corpus-based n-gram analysis are all possible using the introduced methodology. Independently, it will be necessary to select a wider range of malicious programs and even conduct (targeted) attacks on the template system in order to further test its robustness.

In terms of process and event type extraction, the introduced system promises plenty of future applications in need of evaluation. For instance, it becomes possible to easily collect names and paths of malicious processes responsible for triggering undesired activity in the OS. This information can be used to supply conventional host-based IDS with blacklists of malware and to ascertain which processes are most often mimicked by

attackers. Furthermore, certain types of events may be more represented than others when it comes to hostile behavior: For example, further evaluation may bring to light that the presence of image load events is a stronger indicator of an attack than e.g. registry reads.

4.7 Summary

In this chapter, we introduced a methodology for extracting a dictionary of sentiment-rich event bigrams previously recorded at OS kernel level. Unlike similar approaches, we consider each event's process context while maintaining their chronological order. Log likelihood ratio testing is used to determine behavioral cohesion and additionally serves as the foundation for grading bigrams according to their likely maliciousness. The resulting dictionary can be used as a template for scoring the alignment of unknown behavioral sequences and offers a data basis for future classification and anomaly detection as implemented by the AIDIS core system. In summary, this component of our semantics-aware solution contributes by:

- Presenting a smart trace construction method that considers both chronology and process context of abstracted kernel events;
- Introducing a bigram-based technique that determines likely collocations as well as the maliciousness of an event sequence through its log likelihood ratio;
- Transparently compiling a WordNet-like sentiment dictionary of benign and malicious kernel event pairs;
- Implementing a fast scoring approach that combines the advantages of signature- and behavior-based detection.

During our research we compiled two dictionaries – one for our abstracted events and another for raw system and API functions. Both are readily available for future use. Initial scoring results are promising: We achieved an accuracy score of 98.2% (100%, if semantic-neutral traces are discarded) for smart event traces as opposed to 73.0% for conventional function call sequences. Especially in the malware context, our proposed data format has proven to be far more accurate than the tested API call alternative. Thanks to the linear increase of processing complexity, our system can furthermore be deployed on even low-performance machines, provided that the installed amount of RAM exceeds 16GB for a set of 1 million or more unique activity pairs.

The LLR system proved to be a suitable approach to identifying processes that are more likely to be involved in malicious activity. Seven Windows core programs were shortlisted, with `svchost.exe` the most promising candidate for continuous monitoring thanks to its versatile role and the resulting high number of events generated during an average system session. The graph-based system introduced in this thesis uses the process selection made in this chapter for all its anomaly detection and classification tasks.

LLR-based sentiment extraction of bigrams is a viable way of mining knowledge not only from natural language texts, but also from OS activity logs such as kernel event traces and function call sequences. We succeeded in designing a transparent and obfuscation-resistant behavior-based learning system able to shift the focus from sample-centric analysis to ubiquitous OS process analysis. In comparison to the API and function call solutions introduced in the 'Related Work' section (see Section 8.2.1 for details), our system offers improved accuracy and a wider range of features. While no other sentiment-like systems currently exist in the domain of cyber-security to the best of our knowledge, our LLR-based approach scores more than 12 percentage points higher than natural language solutions such as the one presented by Gamon [111], and 4 percentage points higher than the system by Wiebe et al. [341], which seeks to distinguish subjective and objective language. This supports our hypothesis that human language processing techniques can be applied to traces of OS behavior and opens up new research opportunities for similar endeavors.

Chapter 5

Grammar Inference of Anomalous Behavior

Contents

5.1 Introduction

Pattern and anomaly detection systems usually suffer from a lack of semantic interpretation. The so-called *semantic gap*, the hard-to-bridge difference in syntactic event

information and actual attack semantics, remains an issue. Patterns of binary appearance or execution behavior are often manually assigned to represent analyst knowledge, while anomaly detection systems do not usually attempt to explain the identified deviations. This makes potential victims vulnerable to unknown attacks and does little to further the exploration of meaning and intent behind the actions of a malicious actor.

Successfully discovering potentially harmful system behavior boils down to three major problem domains: i) the automated generation of patterns that contribute to detecting and understanding complex multi-stage attacks, ii) attack semantics, and iii) the holistic view on targeted attacks and their many properties. Arguably, a powerful formal definition of malicious behavior is the foundation for addressing most of these aspects.

In this chapter, we propose a novel IT system behavior inference and classification methodology based on the Sequitur algorithm [234], which we formalize through a context-free grammar (CFG) extended by semantic attributes. The approach combines a condensed formal definition with the generation of knowledge linked to the information security and malware analysis domains. Instead of manually defining the many terminals and production rules that the description of a behavior trace would require, we automate the process through an extension of Sequitur that is fully capable of determining and evaluating significant rules. This, together with our output visualization and analytics system, eliminates the analysts' need to define fixed patterns describing harmful or benign behavior.

The remainder of this chapter is structured as follows: In Section 5.2, similar works in the area of security-related inference are reviewed. In Section 5.3, the specifics of grammar inference and available algorithms are discussed. We furthermore introduce our input event data as well as the developed attribute grammar for describing (malicious) system activity, and present our inference algorithm of choice. Applied grammar inference and data analysis procedures are detailed in Section 5.4. Our implementation, which includes a visual knowledge extraction prototype (Sections 5.5 and 5.7), as well as several evaluated applications of the approach (Section 5.6) conclude the chapter.

5.2 Related Work

In light of the large number of operating systems and programming languages currently available, a universal means of abstraction and classification of malicious behavior into a more generic representation is paramount. Jacob et al. [140] present a detection system based on attribute grammars, where syntactic rules describe possible combinations of operations constituting certain behavior, while semantic rules control the data flow between events and assign general meaning to a sequence. Jacob et al's system is intended as formal foundation for developing robust intrusion and malware detection automata. On the modeling side, Filiol et al. [99] propose a generalized model for malware recognition which considers both sequence-based and behavior-based detection.

An evaluation methodology for behavioral engines of existing products is proposed. The major difference to SEQUIN is the system's reliance on manually defined behavioral rules. Jacob et al.'s approach focuses on specified duplication and propagation behavior of investigated PE and VBA samples. Close to 20 Windows native API calls are directly mapped to interaction classes (create, read, write, etc.) and object types such as file, registry, or network operations. SEQUIN's grammar inference automates this process, but relies on an effective naming schema (see 5.4.4) to retain the semantic link.

In general, the discovery of program behavior is key to understanding benign and malicious software. Zhao et al. [357] present a semi-automatic graph grammar approach to retrieving the hierarchical structure of an application's activity. This is achieved by mining recurring behavioral patterns from execution traces using VEGGIE with SubdueGL [20, 19], a Minimum Description Length (MDL)-based compression algorithm. The inferred graph grammar and a syntactic parse tree visually represent reused structures found. Unlike SEQUIN, Zhao et al.'s semi-automated approach uses a more computationally complex context-sensitive grammar to identify common call subgraphs, which are ultimately used in code verification scenarios.

Joo and Chellappa [147] introduce a method for representing and recognizing specific event anomalies in a video by using attribute grammars. This limited domain makes it possible to model most of the events expected to occur and to define anomalies that do not fit the model. Matches are represented by degree of certainty expressed as a probability. While Joo and Chellappa's and our system share the formal foundation, they differ significantly in inference capabilities and automation.

Thompson and Flynn [309] use a similar algorithm to detect and identify the polymorphic instances of a given malware. The approach represents program structure as a context-free grammar, and compares grammars by checking for homomorphism between them. The system makes it possible to identify variants of software by abstracting the control flow of the code. Non-structural elements are removed and the complexity of code quantified by counting the number of remaining elements within the function. For comparison, the resulting grammar is serialized (similar to SEQUIN's zero rule) and checked against another string. Inference mechanisms or CFG attributes for function arguments are not employed. Thompson and Flynn's purely theoretical approach does not specify concrete algorithms or evaluation scenarios.

On the more traditional anomaly detection side, Creech and Hu [66] introduce a host-based detection method that uses discontiguous system call patterns. The authors use a context-free grammar to describe (but not infer) benign and malicious call traces. Several decision engines were tested and compared in the paper, making it a good starting point for the selection of learning algorithms applicable to system call sequences.

In a patent submitted by Eiland et al. [90], the authors describe an intrusion masquerade detection system that includes a grammar inference engine based on MDL compression. The compression algorithm is applied to sets of input data to build user-specific grammars. The use of intrusion masquerade is ultimately based on the

determined distance between template and observed algorithmic minimum sufficient statistic.

Visualization is a predominant theme in this field. With GrammarViz, Senin et al. [282] introduce a grammar mining and visualization tool based on CFG induction. While GrammarViz does not specifically consider attributes or malicious software scenarios in general, it describes a practical approach to manually analyzing time series data. In a more recent paper, Senin et al. [283] expand on the concept of algorithmic incompressibility for anomaly detection and present practical examples using spatial trajectory data. Senin et al.'s Sequitur-based system heavily relies on the ratio between rules and terminals for identifying anomalies. Unlike SEQUIN, the employed visualization tool does not fully support internal pattern extraction and classification while primarily depicting time series charts.

Specific grammar inference algorithms considered during the design stages of SEQUIN are introduced in Section 5.3.1 below.

5.3 Preliminaries

In this chapter, we introduce the grammar inference background of our solution and specify type of system event data used as the foundation of semantic pattern analysis. Sequitur, as our compression algorithm of choice, is used to determine patterns of interest. Furthermore, a formal definition of the information used is presented in the form of an attribute grammar.

5.3.1 Grammar Inference

Grammar inference is the process of automatically learning a grammar by examining the sentences of an unknown language [298]. In the IT sector, grammar inference is primarily used for pattern recognition, computational biology, natural language processing, language design programming, data mining, and machine learning. The effectiveness of grammar inference is influenced by the language class and by the information available about the target language. To raise the effectiveness of grammar inference, a combination of learning modules and language classes are used. Grammar inference has been proven to be a feasible approach to anomaly detection, since "algorithmic incompressibility is a necessary and sufficient condition for randomness" [215]. Simply put, this means that sequences of terminals that are not replaced by rules as part of the induction process represent anomalous (i.e. non-recurring) words within a corpus of text, which can be seen as behavioral outlier in the context of an execution trace. We specifically use grammar inference as key component in the process of compressing event traces to extract relevant patterns and to shorten corpora by replacing repeating sequences with representative non-terminals.

	Supervised	Semi-supervised	Unsupervised
Statistical	Co-training [297]	Self-training [207]	ADIOS [292]
Evolutionary	GA-based [273]		LAgts [39]
Heuristic	ALLiS [74]		Inductive CYK [233] ABL [321]
MDL			e-GRIDS [246] CDC [58] VEGGIE [20, 19] Eiland et al. [90]
Greedy search			ADIOS CDC Incremental parsing [281, 14] Sequitur [234] GraphViz [282, 283]
Clustering	EMILE [3]		CDC

Table 5.1: Grammar inference algorithms and applications by category

The main *issues* of grammar inference are over-specialization and over-generalization. Over-specialization (over-fitting) describes the problem when the inference process produces a grammar whose language is smaller than the unknown target language. This is typically countered by defining an appropriate validation set from the available data and by measuring the performance on this data after each training example has been processed [86]. Over-generalization occurs when the inference process produces a grammar whose language is larger than the unknown language. The use of negligible items results in an unnecessarily large grammar. To limit the impact of over-generalization, it is recommended to also use a set of negative examples.

There are various *computational techniques* suitable for grammar inference. D'Ulizia et al. [86] surveyed various algorithms and categorized them into six groups:

- **Statistical methods** use probability distribution in a class of models derived from empirical data generally provided by a large body of text. Applications include self-training [207] and co-training [297]. ADIOS [292] also uses statistical information to derive regularities from sentences.
- **Evolutionary computing techniques**, often used in computational biology, regularly update (evolve) the initial model or grammar. Each new iteration is produced by removing less desired solutions. GA-based approaches and e.g. LAgts [39] both use genetic algorithms to eliminate unnecessary non-terminal symbols and production rules from the grammar.
- **Heuristic methods** generate training examples of sentences. In grammar inference, ALLiS [74] uses heuristics to reduce the number of similar rules as well as for selecting rules that have the most content. ABL [321] finds, with the help of heuristics, the longest common sequence shared between sentences.
- **Minimum description length** (MDL) [264] assumes that the simplest, most compact representation of data is its best and most probable depiction. The principle finds its primary application in data reduction, where "any regularity in

a given set of data can be used to compress the data" [119]. Examples include CDC [58] and e-GRIDS [246].

- **Greedy search algorithms** make decisions based on their internal logic which may lead to the creation, removal or fusion of rules. For example, Sequitur [234] recursively replaces same-character sequences with new production rules. It produces a grammar that reflects repetitions and thereby infers the hierarchical structure of the grammar. ADIOS [292] also applies a greedy learning algorithm to its graph representation of sentences.
- **Clustering techniques** require a starting grammar that contains all possible sentences. They subsequently cluster syntactic units until the grammar has been constructed. For example, EMILE [3] clusters expressions that occur in the same context, while CDC [58] creates sets of sequences within a context before selecting clusters that satisfy the MDL principle (see above).

Grammar inference algorithms are further classified by their learning approach [86]. Refer to Table 5.1 for an overview of the above-mentioned systems, among others. In Section 5.3.2 below, we further discuss the choice of utilizing a greedy, unsupervised inference approach for our prototype.

5.3.2 Algorithm Selection

We have identified three prerequisites for the successful identification and extraction of interesting behavior from a trace:

- **Unsupervised learning** – The learning module used for generating knowledge from malicious system behavior must be unassisted. Human intervention in the decision of whether a grammar is valid or not would contradict the automation requirements set by most analysts.
- **Context-free grammar** (CFG) – A compromise between a regular and a context-sensitive grammar, CFGs offer a good balance between ease of parsing and computational efficiency. The language created by a CFG can be recognized in $O(n^3)$ time, which will prove helpful in future parsing efforts. This selection prerequisite was complemented by the decision to use an attribute grammar for formal representation (see 5.3.3).
- **Loss-less operation** – It is vital that the algorithm employed does not change the order or immutability of events, since both is likely to have an impact on the semantics of a behavioral sequence.

After surveying numerous algorithms such as the ones listed in Section 5.3.1 and Table 5.1, we decided against using an approach that uses equivalence classes in its vocabulary (like ADIOS [292]). While this feature might be interesting for studying mimicry attacks or the use of equivalent API functions at a later point, the first prototype needed to be precise and deterministic in its inference process in order to enable a more expressive evaluation of the general approach. The choice fell on Sequitur.

Sym	String	Grammar	Remarks
1	a	S → a	
2	ab	S → ab	
3	abc	S → abc	
4	abcd	S → abcd	
5	abcdb	S → abcdb	
6	abcdbc	S → abcdbc	bc appears 2x
		S → aAdA	*bigram uniqueness*
		A → bc	
7	abcdbca	S → aAdAa	
		A → bc	
8	abcdbcab	S → aAdAab	
		A → bc	
9	abcdbcabc	S → aAdAabc	bc reappears
		A → bc	
		S → aAdAaA	*bigram uniqueness*
			aA appears 2x
		A → bc	
		S → BdAB	*bigram uniqueness*
		A → bc	
		B → aA	
10	abcdbcabcd	S → BdABd	Bd appears 2x
		A → bc	
		B → aA	
		S → CAC	*bigram uniqueness*
		A → bc	B used only 1x
		B → aA	
		C → Bd	
		S → CAC	*rule utility*
		A → bc	
		C → aAd	

Table 5.2: Operation of Sequitur [234]. Property application is *italicized*.

Sequitur

Sequitur is a greedy compression algorithm that creates a hierarchical structure (CFG) from a sequence of discrete symbols by recursively replacing repeated phrases with a grammatical rule [234]. The output is a representation of the original sequence, which effectively results in the creation of a context-free grammar. The algorithm creates this representation through two essential properties, which are called *rule utility* and *bigram uniqueness*. Rule utility checks if a rule occurs at least twice in the grammar, while bigram uniqueness observes if a bigram occurs only once. A bigram in this context describes two adjacent symbols or terms. Assuming we have a string `abcdbcabcd`, the first bigram would be `ab`, followed by a second bigram `bc`, and so forth. See Table 5.2 for a complete example of the process.

Sequitur is linear in space and time. In terms of data compression, the algorithm can outperform other designs that achieve data reduction by factoring out repetition. It is almost as performant as designs that compress data based on probabilistic predictions [234].

5.3.3 Formal Definition

To enable the conversion of any kind of trace into an applicable ruleset for behavioral classification, it is necessary to formally define relevant (malicious) actions through distinct patterns that can be integrated into a grammatical hierarchy. Unlike many other solutions (see 5.2), we do not manually map system activity to concrete events but use Sequitur-enabled inference to automatically determine likely rules.

For this purpose, our system uses a context-free grammar extended by attributes [4]. While extendable to a full attribute grammar, we specifically use a CFG over a parametrized alphabet in our definition, where attributes are assigned to terminals only. This decision followed an in-depth review of several grammars and languages, including graph grammars [31], state transition graphs based on NLC [269], trace languages [129], Lindenmayer systems [184], and the aforementioned attribute grammars [4].

Graph grammars (also known as graph rewriting systems) were identified as the main competitor to our final specification. Following the notation introduced by Benteler [31], we define a graph grammar as $GG = (N, T \cup \Delta, P, S)$, where node labels $n \in N$ are non-terminals and $t \in T$ are terminals, while the respective edge labels e are part of alphabet Δ. $S \in N$ describes the start axiom whereas a production $p \in P$ is part of the set of production rules $P = (L, R, C)$, which describe the one-node graph $L \in p$ that replaces graph $R \in p$ using specific embedding rules $C \in p$. The two main issues with graph grammars are the inflexibility of the edge labels, which, unlike the attributes in attribute grammars, are typically limited to one element $e \in \Delta$, as well as the inherent computational complexity of graph rewriting operations/embedding rules: most algorithms require cubic or greater time to complete. Since our system is intended to be used as compression tool for simplified graph data (see 5.6.1), the performance factor was relevant.

In summary, the reason for our choice was grounded in the fact that semantically interesting connections between system events are often expressed by several parameters; parameters, that can be aptly modeled by the attributes of a context-free grammar. Performance and the availability of parsing tools also factored into the decision.

The inferred patterns and, by extension, the full attribute grammar as per our specification, can be defined as follows:

Let $AG = (G, A, R, V)$ be an attribute grammar, where:

- $G = (N, T, P, S)$ is a context-free grammar
  - N... Set of non-terminal symbols (variables)
  - T... Set of terminal symbols (alphabet)
  - P... Production rules
  - S... Start symbol
- A is a finite set of attributes

- R is a finite set of attribution rules (semantic rules)
- V is a finite set of values assigned to an attribute

Every symbol $X \in (N \cup T)$ is assigned a finite set of attributes A_X. The attribute $a \in A_X$ is denoted $X.a$. Every attribute $a \in A_X$ also has a set of values $V(X.a)$. Typically, an attribute a of symbol $X \in (N \cup T)$ that is e.g. assigned the value "0" is denominated as $X.a = 0$.

Our methodology uses attributes to store parameters of system events, such as the names of particular files that are being accessed or IP addresses that are being contacted in the course of a network operation. Attributes are also used to retain the connection to the invoking process of an event. In our case, attributes are used to e.g. represent the name of a file being created and the name of the process triggering that particular operation (see Sections 5.3.4 and 5.5.1 for more examples).

Two frequently used attributes are $a_1 = X.trigger\_name$ and $a_2 = X.element\_name$. The value $v_i \in V(X.a_1)$ identifies the actual name of the observed process responsible for triggering the individual event $X \in (N \cup T)$. Value $v_j \in V(X.a_2)$ denotes the process or file system element the process interacted with.

Raw system events (see 5.3.4) captured by our monitoring agent are processed by Sequitur, which infers a full grammar in accordance to above definitions. Our system is able to depict an arbitrary number of input traces with several attributes $a \in A$. The resulting grammar enables further parsing and semantic analysis. See Section 5.4.2 for more information about the inference process.

5.3.4 Event Data

Like sentiment extraction, SEQUIN is based on event traces defined as descriptions of operating system kernel behavior invoked by applications. Refer to Chapter 4.3.1 for more information about event types, event linking, smart traces, as well as preprocessing.

5.4 Sequitur-based Grammar Inference and Analysis

5.4.1 Preprocessing

Before Sequitur can be used on log files, behavioral traces or other sequential reports describing the activity of potentially malicious programs, the traces need to be reduced to their core components. In this normalization stage we have the choice to either strip away all attributes, or to retain them in an abstracted fashion as part of the set of terminals. As we want to construct a full, semantics-aware attribute grammar, most information is typically kept. We only reduce volatile information such as (user) IDs, memory addresses, and registry paths to a more manageable set of terminals. Names of known system processes and libraries are not modified in any way while

unknown binaries and modules (which are possibly randomly named) are represented by extension-aware placeholders (e.g. `1.txt` or `2.exe`).

In order to compare the impact of different levels of detail and granularity, we defined a total of three input formats. A full example input and output scenario is discussed in Section 5.5.1.

- **Verbose** – This trace format uses full, attribute-enabled events as individual words of the corpus. In verbose mode, the input data is transformed into the following format: `triggering-process,operation,element-name`, which translates to $v_i \in V(X.a_1)$,$t_x \in T$,$v_j \in V(X.a_2)$. For example, a specific file creation operation triggered by the known `explorer.exe` process would be preprocessed into the following textual input format: `explorer.exe,file-create,1.txt`.
- **Reduced** – In this preprocessing mode, we omit attribute a_2 to generate a quick view of the high-level activity exhibited by the processes under scrutiny. Here, v_j is not processed, resulting in a reduced format of `triggering-process,operation`, depicted as e.g. `explorer.exe,file-create`.
- **Granular** – The goal in granular mode is to investigate operations not as single word, but as elementary components. Each of the elements processed in verbose mode is treated by Sequitur as one terminal of the bigram. To maintain a level of separation between event triples, a forth item denoting the start of a new event is prepended before each v_i. This results in the following input (items delimited by semicolon): `<start>;triggering-process;operation;element-name`.

5.4.2 Rule Extraction

Since Sequitur only takes a single input file per default, we added additional functionality to the algorithm in order to retain information of origin and to enable multi-file inference with various evaluation subroutines. This way, SEQUIN's grammar inference can be applied to several files at once without having to concatenate the input in advance. Specifically, we altered Sequitur to be capable of constructing rules across file boundaries denoted by a unique separator, which is ignored by the inference engine. This ultimately enables comparative analyses of larger, disconnected data sets that do not necessarily share repeating behavior within a single trace, which, under normal circumstances, is required for the inference process to trigger. The main stages of the rule extraction process are the following:

- **Lexical analysis** – In this initial step, each unique terminal $t \in T$ is assigned a corresponding symbol, called a token. This numerical representation is used to streamline the process by reducing the processing complexity of string-only comparisons. Each new terminal is additionally stored in a translation (symbol) table for later reference.
- **Grammatical inference** – After the lexical analysis process, the Sequitur algorithm is applied to generate an execution trace grammar consisting of tokenized

terminal symbols. The first rule $p \in P$ of each grammar is the start rule, or 'zero rule', which depicts the full grammar of the compressed input data. In our implementation, every line thereafter contains the following extracted information:

- Rule – The rule consists of a left-side rule name (variable), which is sequentially numbered, as well as right-side variables and terminals. The non-terminals are, again, references to finer-grained rules while the terminals represent the actual system events. In line with the definition of CFGs, there is only one single variable on the left side of a rule.
- Resolved rule – In order to provide a detailed view on individual rules, we recursively resolve each sequence of non-terminals $n \in N$ to their base terminals $t \in T$.

5.4.3 Rule Evaluation

As part of the evaluation process, the final grammar is parsed to determine how many times a specific derivation occurs in each of the investigated input files. Semantically interesting patterns include specific sequences that e.g. occur exactly once in each input trace, making them potential common denominators for a class of malicious behaviors. The computed information includes:

- File rule (FR) count – This number shows how many times a rule occurs in the current input file.
- Grammar rule (GR) count – The overall count across all supplied input files is specified here. For a single trace, this number is identical to the FR count.
- Prevalence count – This value specifies the number of input files a particular derivation has been found in. The result is displayed as x/y (x in y), where x is the number of files the pattern is prevalent and y is the overall count of individual input files.
- Match flag – The extraction of interesting rules is facilitated by determining rules that are identical in occurrence and number across all of the processed input files, indicated by a Boolean flag.
- Rule length – this value defines the overall number of items seen in the entire derivation (i.e. the resolved rule). Multiples of a specific rule length are likely to represent recursively compressed rules, as is apparent for rules 81 to 83 shown in Table 5.4.
- Rule density – this support metric facilitates anomaly detection by calculating the ratio between inferred rules and single terminals that are present in the input as well as rule zero.

The various counts always include references to the original input files, which help retain each pattern's connection to its semantic source. In Section 5.5.1, we show an exemplary scenario for a 'verbose' (see Section 5.4.1) input set.

5.4.4 Rule Transformation

In order to transform the newly inferred rules into an attributed grammar as defined in Section 5.3.3, a set mechanism is required. This mechanism allows the analyst to easily define a naming schema for inferred rules, resulting in a semantic description of the terminals and non-terminals contained within. In the initial version of our tool, we map each operation to an attribute-enhanced terminal while rule identifiers are transformed into descriptive variables: Specifically, each rule is dubbed in accordance to its semantic nature. For example, a rule describing a `process-create` operation followed by a `file-delete` operation is transformed into the descriptive variable CREATE-PROC_DELETE-FILE. A rule that describes the loading of two image files is dubbed LOAD2-IMG.

The full naming schema NS is currently defined as follows:

- $NS = (O, E, MO, ME, L)$, where
  - Operation O = {CREA, MOD, START, LOAD, KILL, DEL, CONN}
  - Event type E = {PROC, THR, IMG, FILE, REG, NET}
  - Operation mapping rules MO = {
    CREA → create,
    MOD → modify | change | edit,
    START → start | spawn,
    LOAD → load,
    KILL → kill | stop | terminate,
    DEL → delete,
    CONN → connect
    }
  - Event mapping rules ME = {PROC → process, THR → thread, IMG → image, FILE → file, REG → registry, NET → network}
- and labeling rules L, where
  - $(O_1 || \text{“-”} \; || E_1 || \text{“\_”}, ..., O_n || \text{“-”} \; || E_n)$
  - If $O_n == O_{n+1}$ then $O_n ||$ “2”

The triggering process and element name are then transformed into the attributes tp ($X.a_1$) and en (a_2). Recursive variable descriptors are supported – above naming schema always considers the fully resolved rule. See Section 5.5.1 for several examples of automatically determined variables.

Future versions of the method will replace the current mapping with a true semantic descriptor that identifies specific attacker actions or objectives. While a manual assignment of such variables is already possible [81], it is not feasible in larger analysis scenarios. The automation of the process is an important research challenge to come.

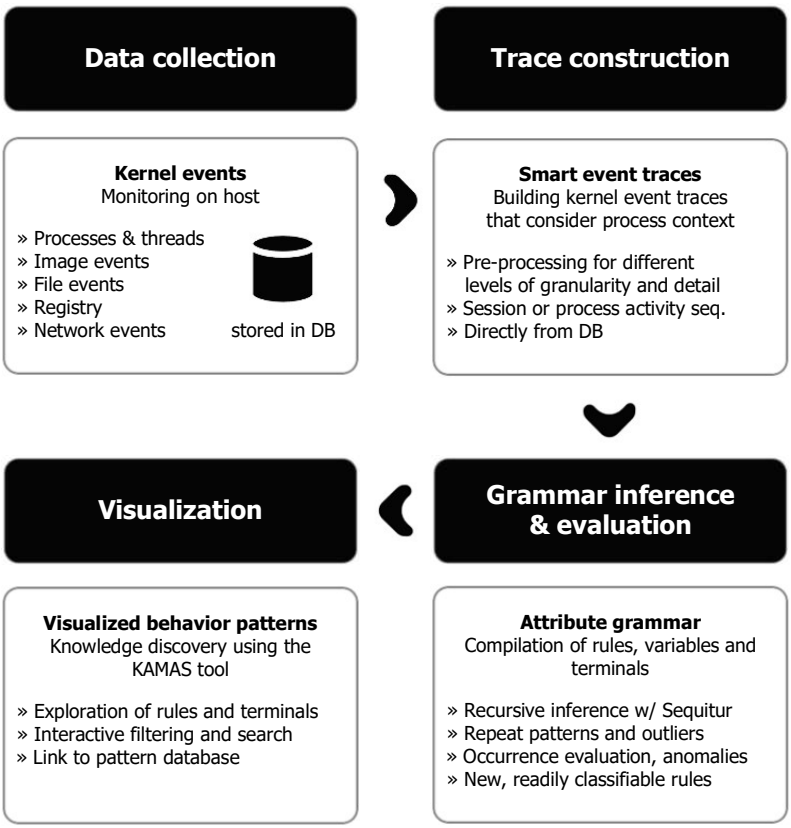


Figure 5.1: Overview of SEQUIN. Following AIDIS' data collection and trace construction, a context-free grammar is inferred from the input.

5.5 Implementation

Our grammar inference and evaluation tool is based on the Sequitur application developed by Eibe Frank[1]. All core and extended functionality has been fully implemented in Java. The data used as basis for the analysis process is collected using a specifically created kernel driver agent deployed on 10 actively used and malware-free (fresh installations performed by security analysts) Windows 7 and Windows 10 machines within our company partner's environment. An additional virtual Windows instance is used for dynamically analyzing malicious software. All machines at least provide common user applications such as Microsoft Office, Adobe Reader, various browsers, as well as common OS extensions such as Java SE and the .NET framework. The collected events are stored and processed on a dedicated PostgreSQL database server that generates verbose or reduced traces of specific processes or even entire system sessions. These traces are ultimately used as input for the Sequitur approach: SEQUIN concatenates the respective files and keeps them apart by inserting a file delimiter that is ignored by the inference engine. This way we can handle an arbitrary number of traces without major changes to the underlying algorithm. Following the inference and analysis stage, our KAMAS prototype visualizes the grammar and helps discover and classify relevant elements. See Figure 5.1 for a full process overview.

[1] https://github.com/craignm/sequitur/tree/master/java

In practical scenarios, it might be prudent to use clustering algorithms to pre-classify traces that are likely to share common behavior. While these algorithms typically do not yield insight into event semantics, this intermediate step helps an analyst to select sequences that e.g. belong to a similar class of malware or describe a comparable attack stage. In such a scenario, our inference tool can be used to specifically extract behavioral patterns for a particular use case. In our initial tests, we used Malheur [263] for this purpose.

5.5.1 Example

With or without preselection, our grammar inference and evaluation tool will generate variables and production rules for a dynamically growing number of terminals and attributes. Below example demonstrates the use of our tool for two simplified 'verbose' input files generated from aforementioned kernel event traces. Thread context information and smart reordering has been omitted for better legibility. The character tokens ($a..m$) were added manually for better understanding.

```
Input file 1: Verbose mode (default sequence). Delimiter: newline.
    explorer.exe,file-create,1.exe (a)
    explorer.exe,process-start,1.exe (b)
    1.exe,image-load,kernel32.dll (c)
    1.exe,image-load,advapi32.dll (d)
    1.exe,registry-modify,hklm/software/microsoft (e)
    1.exe,registry-modify,hklm/software/microsoft (e)
    1.exe,process-create,cmd.exe (f)
    cmd.exe,process-create,net.exe (g)
    1.exe,registry-create,machine/system (h)
    1.exe,registry-modify,hklm/software/microsoft (e)
    1.exe,registry-modify,hklm/software/microsoft (e)
    cmd.exe,process-kill,net.exe (i)
    1.exe,thread-terminate,thread (j)
    explorer.exe,file-delete,1.exe (k)
```

The second input file has been determined by Malheur to be similar, however the commonalities are yet unclear. This is where our pattern evaluation extension comes in.

```
Input file 2: Verbose mode (default sequence). Delimiter: newline.
    explorer.exe,file-create,1.exe (a)
    explorer.exe,process-start,1.exe (b)
    1.exe,thread-create,thread (l)
    1.exe,image-load,kernel32.dll (c)
    1.exe,image-load,advapi32.dll (d)
    1.exe,image-load,ws2_32.dll (m)
    1.exe,registry-modify,hklm/software/microsoft (e)
    1.exe,registry-modify,hklm/software/microsoft (e)
    1.exe,process-create,cmd.exe (f)
    cmd.exe,process-create,net.exe (g)
    cmd.exe,process-kill,net.exe (i)
    1.exe,thread-terminate,thread (j)
    explorer.exe,file-delete,1.exe (k)
```

Sequitur now infers the following rules and evaluates the frequency and similarity. Below output has been reformatted to contain the resolved events only for the remaining, recursively created rules (rule 3), as well as rule zero. Rule density is only calculated for the latter.

Rule 1 is found twice across the grammar (GR count) of the two appended input files. The two events `explorer.exe,file-create,1.exe` (abbreviated: a) and `explorer.exe,process-start,1.exe` (b) form the rule 1 → a b. For a more formal breakdown summary of example input file 1, please consult below grammar depiction AG_1.

```
Rule: 1
    explorer.exe,file-create,1.exe (a)
    explorer.exe,process-start,1.exe (b)
Evaluation:
    FR count (file 1, 2): 1, 1
    GR count: 2
    Prevalence: 2/2
    Match: true
    Rule length: 2
```

Rule 2 is inferred by events c and d: 2 → c d. It is part of both files and is thereby prevalent in the input.

```
Rule: 2
    1.exe,image-load,kernel32.dll (c)
    1.exe,image-load,advapi32.dll (d)
Evaluation:
    FR count (file 1, 2): 1, 1
    GR count: 2
    Prevalence: 2/2
    Match: true
    Rule length: 2
```

Below rule is part of both input files, but does not perfectly match in terms of frequency: The second trace contains two identical occurrences instead of just one. It resolves to 3 → e e.

```
Rule: 3
    1.exe,registry-modify,hklm/software/microsoft (e)
    1.exe,registry-modify,hklm/software/microsoft (e)
Evaluation:
    FR count (file 1, 2): 2, 1
    GR count: 3
    Prevalence: 2/2
    Match: false
    Rule length: 2
```

Prevalent rule 4 (see below) is the only persisting recursively inferred sequence of the example: It translates to 4 → 3 f g. However, Sequitur does not immediately build that rule: Initially, the system infers 4TEMP1 → 3 f, followed by 4TEMP2 → 4TEMP1 g. Because of the rule utility property, both TEMP rules are ultimately dissolved, resulting in the final rule 4 → 3 f g.

The final rule (see below) summarizes the triple that concludes both traces: 5 → i j k. Like rule 4, this process has an intermediate step: Before settling on the final derivation, Sequitur builds the rule 5TEMP1 → i j, followed by 5TEMP2 → 5TEMP1 k.

```
Rule: 4
    1.exe,rule,rule-3
    1.exe,process-create,cmd.exe (f)
    cmd.exe,process-create,net.exe (g)

Evaluation:
    FR count (file 1, 2): 1, 1
    GR count: 2
    Prevalence: 2/2
    Match: true
    Rule length: 3
Resolved:
    1.exe,registry-modify,hklm/software/microsoft (e)
    1.exe,registry-modify,hklm/software/microsoft (e)
    1.exe,process-create,cmd.exe (f)
    cmd.exe,process-create,net.exe (g)
```

```
Rule: 5
    cmd.exe,process-kill,net.exe (i)
    1.exe,thread-terminate,thread (j)
    explorer.exe,file-delete,1.exe (k)
Evaluation:
    FR count (file 1, 2): 1, 1
    GR count: 2
    Prevalence: 2/2
    Match: true
    Rule length: 3
```

From these 5 final rules, a compressed zero rule can be inferred. The resolved rule yields the concatenated original input of both files separated by a file delimiter: 0 → (file 1) 1 2 4 h 3 5 (file 2) 1 l 2 m 4 5.

```
Rule: 0
    explorer.exe,rule,rule-1
    1.exe,rule,rule-2
    1.exe,rule,rule-4
    1.exe,registry-create,machine/system (h)
    1.exe,rule,rule-3
    cmd.exe,rule,rule-5
    -
    explorer.exe,rule,rule-1
    1.exe,thread-create,thread (l)
    1.exe,rule,rule-2
    1.exe,image-load,ws2_32.dll (m)
    1.exe,rule,rule-4
    cmd.exe,rule,rule-5
Evaluation:
    Rule density (input: 3 out of 27): 88.9
    Rule density (rule 0: 3 out of 9): 66.6
Resolved:
    (see concatenated input)
```

In our example, the tool has successfully extracted rules that describe behavior observed in both input files. With the uncompressed events `1.exe,registry-create,machine/system`, `1.exe,thread-create,thread`, and `1.exe,image-load,ws2_32.dll` highlighted, we can immediately spot the deviations from the otherwise recurring behavior.

In conclusion, the output is transformed into an attribute grammar as described in Section 5.3.3. Since semantics is a major factor of rule construction, we assign variables based on the nature of the inferred event. Specifically, above example (here: only for file 1 of rule 0) can be formalized into a grammar as follows:

Let $AG_1 = (G_1, A, R, V)$ be an inferred CFG extended by attributes, where:

- $G_1 = (N, T, P, S)$, and where:
  - N = {CREA-FILE_START-PROC; LOAD2-IMG; MOD2-REG_CREA2-PROC; MOD2-REG; KILL-PROC_KILL-THR_DEL-FILE}
  - T = {
    file-create.tp, en = explorer.exe, 1.exe;
    file-delete.tp, en = explorer.exe, 1.exe;
    process-create.tp, en = explorer.exe, 1.exe;
    process-create.tp, en = 1.exe, cmd.exe;
    process-create.tp, en = 1.exe, net.exe;
    process-kill.tp, en = cmd.exe, net.exe;
    image-load.tp, en = 1.exe, kernel32.dll;
    image-load.tp, en = 1.exe, advapi32.dll;
    registry-create.tp, en = 1.exe, machine/system;
    registry-modify.tp, en = 1.exe, hklm/software/microsoft;
    thread-terminate.tp, en = 1.exe, thread;
    }
  - P = {
    ZERO-RULE → CREA-FILE_START-PROC LOAD2-IMG MOD2-REG_-CREA2-PROC registry-create.tp, en = 1.exe,machine/system MOD2-REG KILL-PROC_KILL-THR_DEL-FILE;
    CREA-FILE_START-PROC → file-create.tp, en = explorer.exe,1.exe process-create.tp, en = explorer.exe,1.exe;
    LOAD2-IMG → image-load.tp, en = 1.exe,kernel32.dll image-load.tp, en = 1.exe,advapi32.dll;
    MOD2-REG → registry-modify.tp, en = 1.exe,hklm/software/microsoft registry-modify.tp, en = 1.exe,hklm/software/microsoft;
    MOD2-REG_CREA2-PROC → MOD2-REG process-create.tp, en = 1.exe,cmd.exe process-create.tp, en = cmd.exe,net.exe;
    KILL-PROC_KILL-THR_DEL-FILE → process-kill.tp, en = cmd.exe, net.exe thread-terminate.tp, en = 1.exe,thread file-delete.tp, en = explorer.exe, 1.exe
    }
  - S = {ZERO-RULE}
- A = {tp; en}
- R is described as part of the preprocessing stage and defines which portion of the data translates into triggering process tp (v_i), operation (t_x), and element en (v_j).
- V = {explorer.exe; 1.exe; kernel32.dll; advapi32.dll; cmd.exe; net.exe; machine/-software/microsoft; machine/system; thread}

Above attribute grammar for part 1 of the zero rule has been generated automatically and can now be used as the foundation for further (attribute-based) parsing efforts.

Scenario	Evaluation	Significance
Compression	Section 5.6.1	Reduction of input data size
		Reduction of processing complexity (third-party data)
Anomaly detection	Section 5.6.2	Detection and extraction of deviating behavior
Baselining	Section 5.6.2	Identification of common patterns in traces
Visualization	Section 5.7	Visual presentation of inferred rules
Discovery	Section 5.7	Interactive filtering and extraction of terminals/rules
		Rule labeling, storing
		Highlighting of known rules

Table 5.3: Scenarios covered by experiments, in order of appearance by subsection

The inferred variables, if stored, can be used as new behavioral templates for comparable input data sets. Above definition can easily be extended to encompass specific files, or all of them at once (entire zero rule).

The next section discusses practical applications of this approach and evaluates them using real-world data.

5.6 Evaluation

The introduced system has a wide variety of applications. Ranging from preliminary knowledge extraction in malware analysis scenarios to understanding more complex attacks, the adapted inference methodology is versatile in both terms of input data as well as practical benefit. Below, we introduce and evaluate some of its applications and discuss future and ongoing work. Table 5.3 provides an overview of the conducted experiments.

5.6.1 Preparatory Data Reduction

Concept

In many malware and APT attack stage analysis scenarios, analysts are often forced to deal with huge amounts of data. Be it kernel events, raw system calls or even assembler-level CPU instruction information, the abstraction and reduction of input data is essential to decrease the complexity of many an analysis task. Our solution provides the means through its easily adaptable preprocessing mechanism (see Section 5.4.1) and the grammar inference system itself. By using the Sequitur approach, it is possible to reduce the input corpus to only relevant n-grams ($n \geqslant 2$), instead of working with the full, unfiltered set of event or code snippet unigrams. The grammar transformation mechanism (see Section 5.4.4) also enables us to work with an automatically generated placeholder variable $n \in N$ instead of several compound terminals.

A second, closely related application of Sequitur compression is the extraction of recurring patterns. Dubbed ‘baselining’, this process takes the result of the inference process and regards sequences of terminals that have been turned into rules. The discovery process is best supported by our KAMAS VA prototype introduced in Section 5.7.

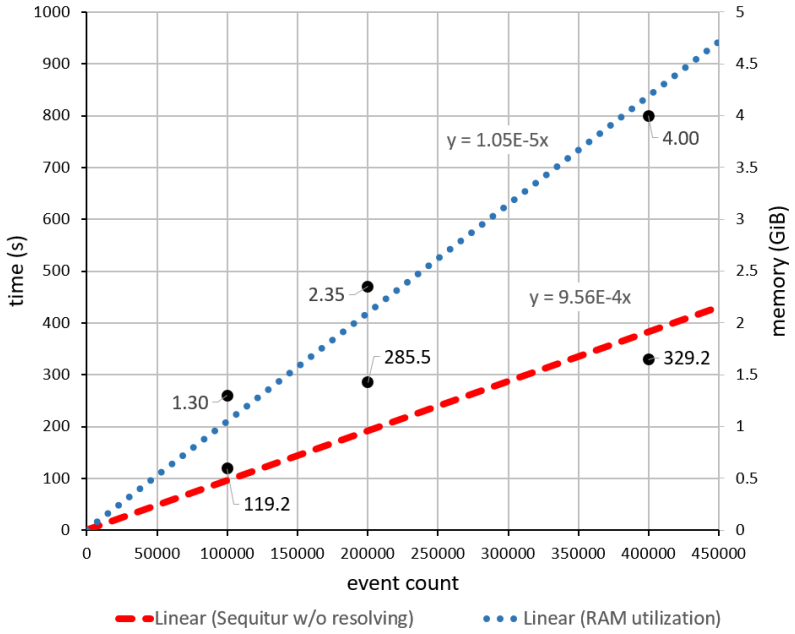


Figure 5.2: Performance analysis of Sequitur inference. Both processing time and RAM utilization grow at a linear rate. The alphabet size (not pictured) was determined to have a greater impact on the time required than on physical memory use.

Evaluation

Current efforts include the pre-abstraction of behavioral data in graph notation (also see Section 5.3.4) subsequently used for edit distance calculations [191]. Minimizing the amount of data to be processed drastically reduces computation requirements of expensive (up to exponential complexity, depending on the application) graph transformation operations. Specifically, we evaluated several days' worth of benign system events monitored by our kernel driver (see Section 5.5 for event capture details), collecting 100k, 200k, and 400k sequential events with different size alphabets and rule density – all associated with instances of the Windows `svchost.exe` process. Under normal circumstances, this data would have to be assessed in its uncompressed entirety, as it is used for creating baseline graph templates utilized in behavior deviation analysis using a combination of Hungarian distance computation [167] and Malheur heuristic clustering [263]. Thanks to our Sequitur-enabled data reduction, we can focus on event sequences (rules) that are representative for specific processes – or on the remaining terminals which constitute a potential anomaly. Both significantly speed up all involved, polynomial complexity star graph matching operations (Figure 5.3) used by the tested system [191]. At the same time, we increase the accuracy of the template creation process by drastically reducing the number of empty feature vectors that are normally produced in the clustering stage.

In our first, medium-sized exemplary dataset of 100k Windows kernel events (alphabet of 665 words), we reduced the number events to a list of 1,715 terminals and 6,415 first-level rules (contained in the zero rule), which effectively compressed the data by 90.7%. This cut the processing time for graph template generation and graph trans-

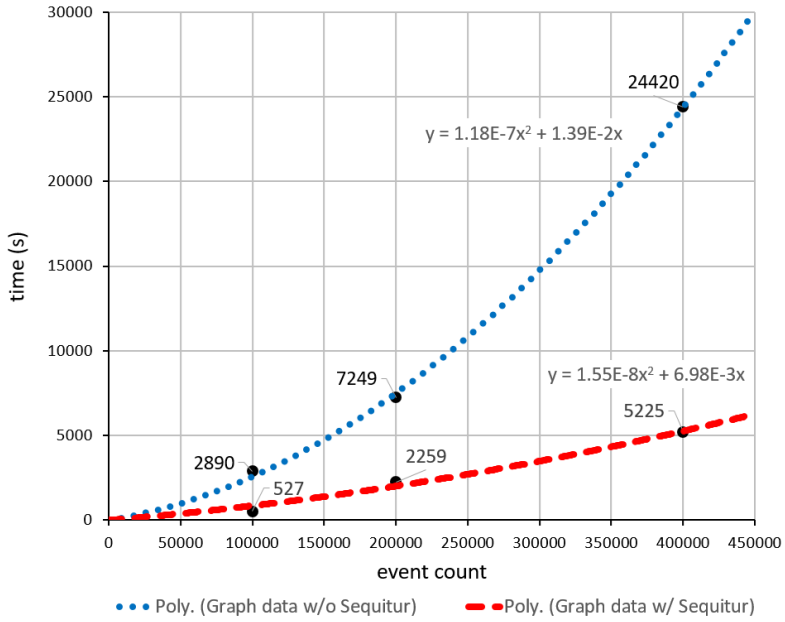


Figure 5.3: Performance impact on graph matching operations. The chart compares the time required for creating a template using uncompressed data versus the non-terminals inferred by Sequitur. Input filtered to rules with a prevalence count $PC > (n/m)$, where $m < 0.2 * n$ (here: $m = 1000$) and $n =$ number of unique processes per dataset. Results show an average speed-up of around 73% for kernel event data.

formation calculations by 77.64%, down to a total of 8.75 + 1.9 minutes, instead of 48.2 minutes sans compression. Performance evaluation showed a maximum memory utilization of around 1.3 GiB, with a total processing time of 1.9 minutes (9.1 minutes with full rule resolving) on a dual core virtual machine equipped with 64 GiB of RAM.

The second and third datasets encompassed an alphabet of 1,310 and 1,569 words, respectively. We achieved a compression rate of 95.95% and 98.26%, which reduced the graph processing time by 64.91% and 77.26%. RAM usage increased to 2.35 GiB for the 200k dataset, and to 4 GiB for the 400k trace. Figure 5.2 and 5.3 show basic regression analyses of grammar inference and graph processing times as well as RAM utilization during compression. See [191] for more information on star graph template generation and matching.

Discussion

The overall process was determined as scaling at linear time $O(n)$, putting it in line with e.g. basic search algorithms and confirming the results disseminated by [234]. RAM consumption scaled linearly as well – in case of our machine, we expect to hit the memory ceiling of 64 GiB at around 6 million events, provided the size of the alphabet grows at a similarly steady rate. SEQUIN without rule resolving performed best overall, saving up to 90% of its processing time for the largest (400k) dataset. Because of this significant overhead, rule resolving will in the future be performed independently from compression – directly in our growing database of known productions. This will offer a convenient

way to compute and look up patterns without negatively impacting performance during the actual inference process.

In terms of semantic accuracy, over-zealous recursive compression needs to be considered when applying the approach to data with large repeating sequences of terminals, as it may combine too many individual events of significance into one long rule, which in turn consists of other lengthy rules describing similar behavior. This could not only skew the result, but also eliminate some of the performance benefits evaluated above. In our experiments, we have found it prudent to impose a limit on maximum resolved length for terminal-only rules. Another approach is to set a minimum limit on prevalence count (see Section 5.4.3) to remove non-terminals that occur in only a few input files. The thresholds for these operations largely depend on the nature of the data used and will have to be determined by experimentation on a case-to-case basis.

SEQUIN has proven to be well suited to the task of preprocessing/reducing input data needed by other, more expensive algorithms. We achieved an average speed-up in star-graph data processing of 73% when employing our system to the same dataset. Data sizes were reduced by up to 98%.

5.6.2 Anomaly Detection

Concept

In our above preprocessing example, we use grammar inference to determine interesting repeating patterns that are representative of the corpus under investigation. However, the reverse is also a viable scenario: By focusing attention on patterns that do not excessively reoccur, our approach can be used to identify anomalies in a sequence or set of sequences. Parts of the trace that are not replaced by variables during rule construction (i.e. the remaining terminals in between) represent unique events that, in such a scenario, are of particular interest as they represent deviating (abnormal) behavior. Rule density (see 5.4.3) is also important in scenarios where stable behavior is expected: the higher the share of terminals, the higher the overall entropy, and, by extension, the likelihood of anomalous behavior. All anomaly detection efforts can be aided by visualization tools such as GrammarViz [282] as well as our own VA research introduced in Section 5.7 below.

Evaluation

The Sequitur tool is not limited to system events but can be used with a wide range of sequential input data formats. In the following, we specifically evaluated an APT anomaly detection scenario on a set of temperature, speed, and photoelectric sensor data generated by a Siemens Simatic industrial control system (ICS) within a testbed environment. We assessed 13 full production runs in total, whereas two of the runs

File	Rule	FR #	GR #	Prevalence	Length
Ben-2	3→139 139	2	2	1/12	16
Ben-2	4→140 140	3	3	1/12	4
Ben-2	12→1-0-1-(...)-0-40 1-0-1-(...)-0-40	2	2	1/12	2
Ben-2	23→1-0-1-(...)-1-56 1-0-1-(...)-1-56	2	2	1/12	2
Ben-2	69→1-0-1-(...)-5-59 1-0-1-(...)-5-59	2	2	1/12	2
Ben-2	98→1-0-1-(...)-8-60 1-0-1-(...)-8-60	2	2	1/12	2
Ben-2	102→0-0-1-(...)-8-54 0-0-1-(...)-8-54	2	2	1/12	2
Ben-2	139→4 4	2	2	1/12	8
Ben-2	140→0-0-0-(...)-0-20 0-0-0-(...)-0-20	2	2	1/12	2
Mal-1	57→1-0-1-(...)-4-60 1-0-1-(...)-4-60	2	2	1/12	2
Mal-1	72→1-0-1-(...)-6-52 1-0-1-(...)-6-52	2	2	1/12	2
Mal-1	81→82 82	2	2	1/12	64
Mal-1	82→83 83	3	3	1/12	32
Mal-1	83→157 157	3	3	1/12	16
Mal-1	84→85 85	3	3	1/12	4
Mal-1	85→1-0-1-(...)-7-60 1-0-1-(...)-7-60	3	3	1/12	2
Mal-1	86→1-0-1-(...)-7-59 1-0-1-(...)-7-59	2	2	1/12	2
Mal-1	157→84 84	2	2	1/12	8

Table 5.4: Extracted and evaluated rules for ICS sensor data traces with low rule density ($\leqslant 40\%$) and prevalence count ($= 1$). Each rule describes an anomaly not typically seen in other input data. FR...file rule, GR...grammar rule.

Sample trace	TRR	TRR*	Length	Length*	Preval.	Preval.*	**Overall**
Ben-1	58.60	0.33	8.87	0.44	2	0.20	0.274
Ben-2	**64.00**	**1.00**	**9.28**	**0.95**	**9**	**0.90**	**0.945**
Ben-3	59.30	0.41	8.9	0.48	2	0.20	0.313
Ben-4	56.00	0.00	8.51	0.00	1	0.10	0.050
Ben-5	56.80	0.10	8.52	0.01	1	0.10	0.091
Ben-6	58.30	0.29	8.79	0.35	2	0.20	0.250
Ben-7	59.20	0.40	8.69	0.22	0	0.00	0.182
Ben-8	60.00	0.50	8.72	0.26	4	0.40	0.426
Ben-9	**63.80**	**0.98**	**9.32**	**1.00**	0	0.00	0.490
Ben-10	57.30	0.16	8.6	0.11	3	0.30	0.226
Ben-11	58.50	0.31	8.79	0.35	**10**	**1.00**	0.660
Mal-1	**62.80**	**0.85**	9.01	0.62	**9**	**0.90**	**0.852**
Mal-2	**62.74**	**0.84**	8.99	0.59	**9**	**0.90**	**0.846**

Table 5.5: Scores for TRR, mean rule length (Length), and rules with minimum ($= 1$) prevalence count (Preval). Columns marked with an asterisk (*) mark values normalized to 0..1 as per Equation (5.1). Normalized scores $\geqslant 0.8$ are printed in bold.

were maliciously altered by illegally interfering with the rotation. This resulted in some atypical sensor readings that are nigh impossible to spot manually.

The full evaluated grammar for a total of 34,000 observed events was constructed within 5 seconds. Sequitur inferred a total of 2,155 rules (sans zero rules), resulting in a 93.7% data compression rate. In stage one, anomaly detection was conducted by assessing rules with a low rule density value. By that metric alone, it was already possible to identify anomalous traces. With a terminal-to-rule ratio (TRR) of over 62.7% (rule density of 37.3%), the malicious samples contained less uniform behavior patterns than the remainder of 11 traces with a mean ratio of 59.3%. Only two benign traces came close to that number, exceeding a TRR of 60%. The comparatively small

margin is due to the fact that, in this scenario, anomalous data did not cause sensor spikes but rather triggered a slow, continuous change in behavior.

Further analysis of the possibly deviating behavior was (and is typically) required to solidify the initial verdict. To this end, we used our evaluation system to filter rules that are present in only a minority of files and that have a prevalence count of 1 out of 12. Armed with the pre-selection based on rule density, we particularly focused on traces with a TRR of >60%. Specifically, we normalized the scores for TRR (x_T), mean rule length (x_L), and the count of low-prevalence rules (x_P) and computed a weighted total X_{WT}:

$$X_{WT} = 0.4\frac{x_T - min(x_T)}{max(x_T) - min(x_T)} + 0.1\frac{x_L - min(x_L)}{max(x_L) - min(x_L)} + \\ + 0.5\frac{x_P - min(x_P)}{max(x_P) - min(x_P)} \quad (5.1)$$

Table 5.5 lists the computed values for each trace. As a result, the Youden index BC_a bootstrap confidence interval [88] was determined as 0.55..1, resulting in an associated criterion score threshold of 0.66 for distinguishing true from false matches. For ICS data, a manual adjustment to a higher threshold (e.g. 0.75) can further increase result confidence.

Now, we can now analyze the outcome of the inference process in detail: Benign trace 2 ("Ben-2" in Table 5.4) contained 9 rules that were not seen in the remaining corpus. Likewise, both malicious traces (pictured here: "Mal-1") contained 9 unique rules. A direct comparison of the remaining anomalous candidates highlights one particularly interesting, recursively compressed block per trace, which resolved into 16 (benign) and 64 (malicious) terminals, respectively. It stands to note that patterns with a higher average length are particularly interesting as they identify larger, uninterrupted sequences unique for the dataset under scrutiny.

See Table 5.4 for a direct comparison of two of the most deviating behavior traces. Rule 3 of "Ben-2", which resolves into 8 terminal pairs as inferred by rule 140 (a rare, but valid sensor state), contributes most to the trace's analysis verdict. For "Mal-1", the same applies to rule 81 (32 iterations of rule 85), effectively identifying the anomalous sensor state. The accuracy of this evaluation scenario is detailed in Figure 5.4 and boasts a high false positive and false negative rate, especially when removing faulty runs. The extracted, semantically relevant rules can now be formalized and stored for future parsing efforts.

For a simple baselining experiment, we used our SEQUIN tool on two lengthy traces depicting the Windows 7 update process performed on two machines with the same OS patch level. With 10,769 events in update 1 and 11,057 events in the second update, the combined compression generated 250 rules in total, 218 of which were contained in both input files. The randomly generated 32-bit strings used as names for the

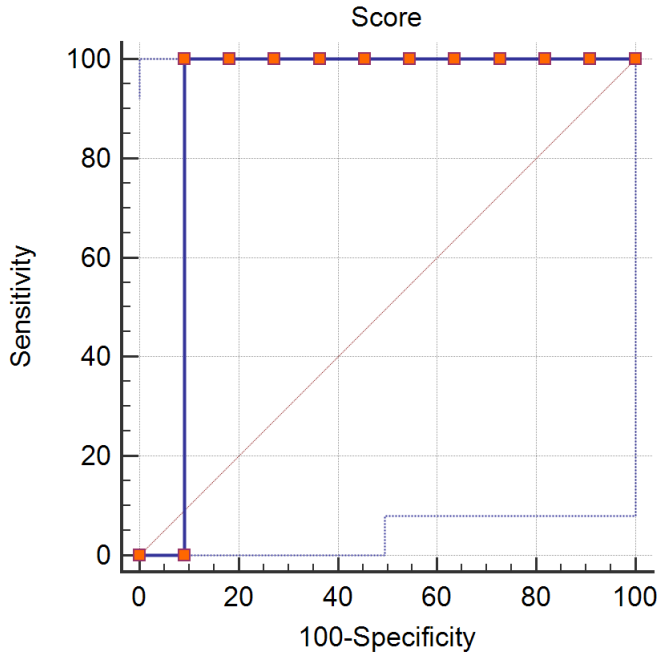


Figure 5.4: ROC curve of the anomaly detection run. 12 out of 13 sets of sensor data traces were classified correctly with a sensitivity (true positive rate) of 100% and a specificity (true negative rate) of 100%. ROC area (AUC) was determined as 0.909. Since the deviating benign run is actually faulty despite not being a deliberate attack, its removal would boost the overall accuracy to 100% in our particular test case.

respective download directories within the `Windows/SoftwareDistribution/Download` folder structure were replaced during normalization as part of the preprocessing stage (Section 5.4.1). The remaining unique rules as well as terminals were deviations from the computed baseline caused by random temporary file names and minor changes in update chronology. In practice, this variant of the inference process can be used to create whitelists, templates for benign process behavior, or, again, for extracting anomalies from seemingly benign sequences of application behavior.

Discussion

In terms of accuracy, anomaly detection or baselining efforts are less likely to require fine-tuning than the compression routine (see 'Discussion' in Section 5.6.1), as the number of recursive rule-building iterations does not negatively impact the result. Instead of limiting the length of resolved rules, the extraction of relevant data is based on choosing the correct rule density and prevalence for the dataset under investigation. Here, analysts need to keep in mind that choosing the latter is tied to the expected number of malicious input traces that might share the same characteristics (identical inferred rules): The more often harmful sequences are assumed to repeat in the corpus, the higher the prevalence threshold needs to be and the less likely it becomes to spot the behavior using anomaly detection.

Initial tests determined an accuracy of well above 92% for the detection of deliberate attacks. Relevant anomalies were found in all of the inspected cases. For more complex scenarios it is recommended to apply visual analytics techniques that enable interactive data exploration: After the extraction of possible anomalies or baselines, a domain expert can investigate further to determine the individual events t that truly describe an illegal action. While this can be done textually using only our inference system, a visuals-assisted solution such as KAMAS promises (see 5.7 below) even better results.

5.7 Visualization & Knowledge Discovery

5.7.1 Visual Analytics

One of the major applications of our proposed solution is undoubtedly the extraction of new domain knowledge. Inferred patterns can be compiled into a permanent grammar used to detect similar behavior in unknown traces. This process is supported by interactive visualization to drastically improve usability. This area of research, typically referred to as visual analytics (VA), forms the basis for KAMAS, our novel system used to visualize SEQUIN's output for pattern discovery and semantic annotation. In the following, we briefly explain the concept of VA, KAMAS' design considerations, and the prototype implementation itself.

VA is "the science of analytical reasoning facilitated by interactive visual interfaces" [308]. A major tenet of VA is that analytical reasoning is not a routine activity that can be automated completely [338]. Instead it depends heavily on the analyst's initiative and domain experience, which is exercised through interactive visual interfaces. Such interfaces, especially information visualizations, are high bandwidth gateways for the depiction of structures, patterns, and connections hidden in the data. Furthermore, visual analytics often involves automated analysis methods that perform various computations on potentially large volumes of data.

When analysts solve real world problems they typically have vast amounts of complex and heterogeneous data at their disposal, as is evidenced by above application scenarios (see Sections 5.6.1 and 5.6.2). Externalization and storing of implicit knowledge will make it available as *explicit domain knowledge*, which is defined as knowledge that "represents the results of a computer-simulated cognitive process, such as perception, learning, association, and reasoning (...)" [48].

Through visualization, explicit knowledge can be used to graphically summarize and abstract a dataset. Put simply, it enables quicker and more precise analyses of complex input data such as the set of traces used in our ICS example.

Using VA for security applications is a widely accepted practice. In Wagner et al. [331], the authors surveyed tools for behavior-based malware analysis in addition to visual representations best suited to various domain challenges. Through a data–users–

tasks analysis [212], they ascertained that the parse tree of a grammar such as the one generated by the Sequitur algorithm, can be abstracted to a directed acyclic graph, where each node represents part of a sequence.

5.7.2 Visualization Considerations

The nature of the event data and the inference algorithm employed builds the foundation for our knowledge-assisted malware analysis tool, dubbed KAMAS [332], which is intended to support analysts in their task of identifying relevant behavioral patterns. In preliminary research [331], problem characterization and abstraction [280] was performed to elaborate the analysts' needs when using visual analytics for behavior-based malware analysis. This way, a common terminology describing i) the data to be visualized; ii) the users of the system; and iii) the tasks to be fulfilled, was established. Based on the outcome of this problem characterization, the initial design decisions for the visualization of data generated by the Sequitur algorithm can be established:

- **Representation of explicit knowledge**. To support the analysts in their task of behavior-based malware analysis, explicit expert knowledge should be made available in the system. Additionally, the actual generation of explicit knowledge needs to be facilitated. For the visualization of that knowledge, a basic hierarchy of events is required. We utilized the malicious behavior schema by Dornhackl et al. [81]: Using the provided semantic categorization, it is possible for analysts to explore currently stored knowledge and to add newly inferred rules to the system. The visualization of the malicious behavior schema employs a tree structure, where the nodes are the different types of malicious behavior and the leaves are the rules for its representation (see Figure 5.5:1).
- **Representation of events**. For the representation of the events included in an analysis file, two important aspects have to be covered. On the one hand, the name of the event (see Section 5.3.4 for more information on input data) is essential for the analyst who is trying to ascertain its purpose. On the other hand, it is very important to learn how often a single event is included in the analysis file. We use a table structure for the visualization of this data, whereby the event name is represented as string and its occurrence is represented as a bar chart including the total number as an overlay. Employing this visualization technique, the analyst gains the ability to quickly find events of interest (e.g., by visually analyzing the size of the bar charts) (see Figure 5.5:3).
- **Representation of rules**. Since a rule is a sequential structure containing several events, it is prudent to use a similar representation as for individual items. In contrast to the representation of all events included in a rule (resolved rule, see 5.4.2), a more abstract visualization can be applied here. The transformation of events based on their unique ID into a graphical representation (which is called 'Graphical Summary' [332]) helps to more effectively locate unknown patterns in the data. Additionally, all the other related information can be visualized as bar

charts in combination with a label representing the total number (e.g., the rule's prevalence and length as introduced in Section 5.4.3). The original order of events within a rule is highlighted (see Figure 5.5:2).

To determine analysts' requirements for behavior-based malware analysis with regards to usable visualization metaphors [331], a basic set of well-known visualization techniques was evaluated. Most of the participants indicated a combination of Multiple View [117], Arc Diagrams [337] and Wordtree [336] as being the most helpful, followed by OutFlow [345] and Pixel-Oriented Visualization [155]. In contrast, the Parallel Tag Cloud [60], has been described as the least complementing solution in respect to behavioral trace analysis. In the following, we detail the visualization concepts leading up to the development of research prototypes for both data processing and visualization.

5.7.3 Implementation

Aforementioned visualization considerations were used as a guideline to define the design rationale of KAMAS [332]. We decided to use an interface concept akin to well-known programming IDE interfaces (e.g. Eclipse) as the conceptual foundation for our prototype. This helps create a familiar environment based on multiple, vertically separated views (see Figure 5.5).

In respect to these interface structures, we placed the tree view of the 'Knowledge Database' (KDB) on the left side of the window (see Figure 5.5:1). The KDB element contains all the explicit knowledge used for automated analysis; newly extracted patterns can be stored in a database for later use, whereas existing ones serve as real-time filter that automatically highlights known patterns. The key 'Rule Explorer' element is positioned in the center of the screen, where most user actions are performed during the analysis of a trace (see Figure 5.5:2). Here, the analyst can see the different rules generated by Sequitur, including all information pertaining to its file and grammar count as well as its length. Selecting a specific element expands the fully resolved rule for further study. Additionally, an arc diagram will be shown to highlight events that constitute a known sequence. The 'Call Exploration' area is positioned on the right side of the interface (Figure 5.5:3), similar to the functions overview area in commonly used programming IDEs. Here, the analyst is provided with the ability to explore all of the events included in the rules that are presented/highlighted in the center element. The 'Call Exploration' view lists all events contained in the loaded file(s). To cope with the potentially huge amount of data, the analyst has the ability to use different, regular-expression-enabled filters to locate data of interest. Newly discovered patterns can be added into the KDB via a simple drag and drop action.

In addition to the IDE-like design decision, we used the Gestalt principles of proximity and similarity [145] to improve interface clarity. Each exploration area (Rule Explorer and Call Explorer) is expanded with its own filtering elements located directly below the visualized data. Based on all aforediscussed findings and concepts, we finally created the KAMAS VA prototype for visualizing and assessing the patterns generated

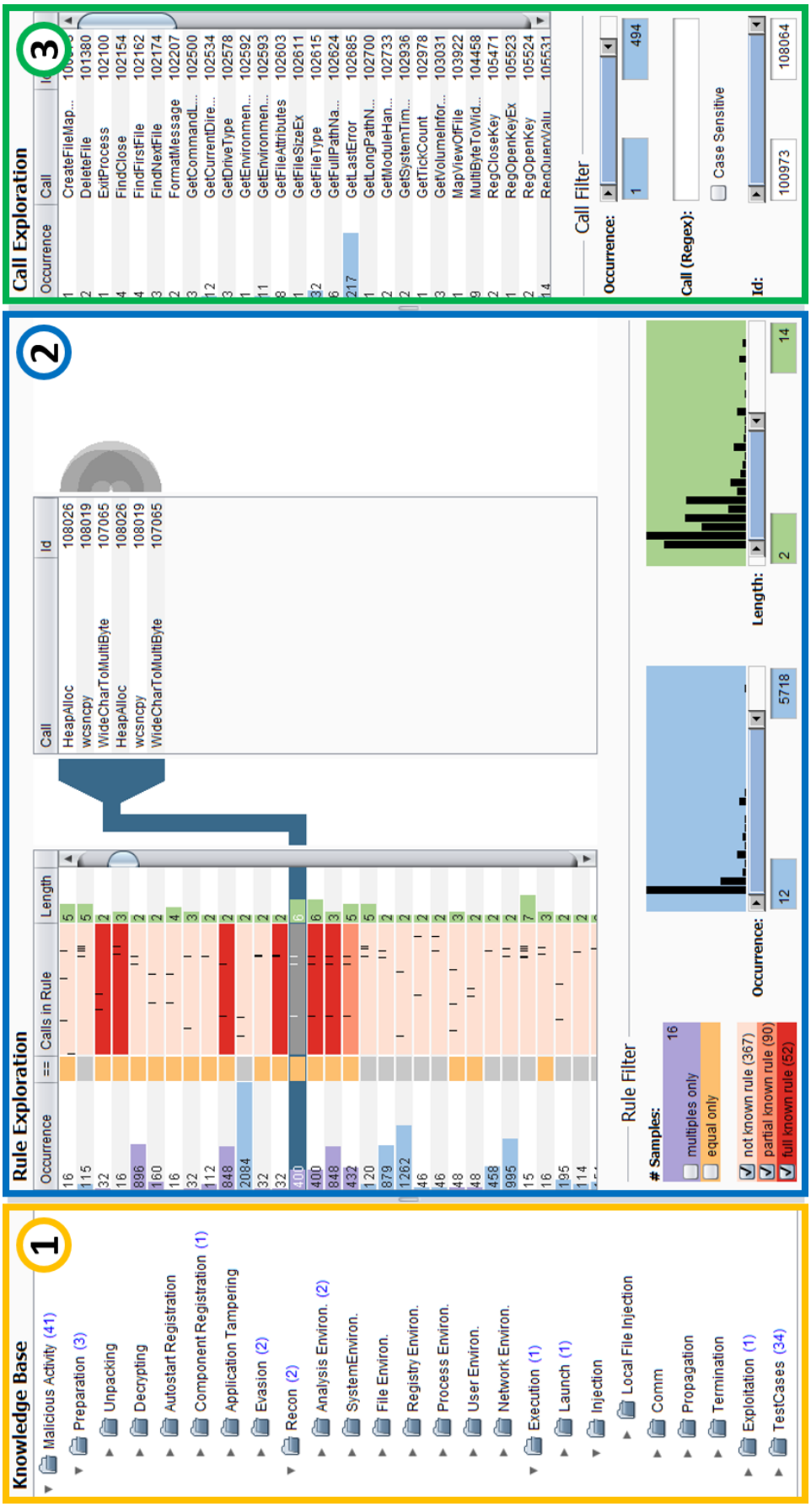


Figure 5.5: Illustration of the Knowledge-assisted Malware Analysis System (KAMAS) designed to support malware analysts in their work. The interface encompasses a (1) 'Knowledge Base' for automated analysis and knowledge sharing between the analysts, a (2) 'Rule Exploration' area, and a (3) 'Call Exploration' area used to investigate individual events. Various filters at the bottom help to remove redundant data.

by SEQUIN. Figure 5.5 depicts a screenshot of the main interface. Analysts have found the tool a useful addition to their workflow of exploring potentially harmful traces of events, be it system calls or OS kernel operations. A full evaluation of KAMAS, including a comprehensive usability study, is disseminated in [332]. The prototype has been released online under Creative Commons Attribution license[1].

5.8 Discussion

While SEQUIN is a versatile solution that offers support to IT security experts and (malware) analysts alike, it is constrained by a number of limitations. As suggested in the discussion of Sections 5.6.1 and 5.6.2, the system does not currently provide an automated way of determining the optimal length of rules representative for an anomaly. The recursive nature of operation of the Sequitur algorithm is likely to produce rules within rules that, semantically speaking, amount to the same result. Right now, it is up to the analyst to impose length, TRR, and prevalence thresholds to counter this limitation.

On the data input side, we are currently required to internally concatenate the individual files and separate them with a file delimiter, which is skipped by Sequitur's rule building engine. This puts greater emphasis on the selection process of input traces, since SEQUIN will only start to infer rules that occur at least twice in the overall input. Too diverse an input will result in a lower compression ratio and anomaly detection accuracy. At the same time, the use of smart traces sacrifices some of the trace's chronology by reordering events according to their process and thread context. While this usually improves the results, long-running processes of lengthy sessions will skew the timeline more than processes/threads that execute, run, and terminate within a shorter window. Here, a mechanism of intelligently segmenting traces will have to be developed to improve accuracy.

One of the other main areas of future improvement is undoubtedly the automated semantic interpretation of inferred variables, which, in the tool's current iteration, are assigned based on the operations (terminals) that constitute the respective rule. In the future, this representation will be updated to include actual attacker goals and (malicious) actions that go beyond the purpose-neutral label currently assigned by the naming schema. Ultimately, we plan to link SEQUIN to a targeted attack ontology linking TAON [188] and the PenQuest model, which already provides the means for labeling data by its attack pattern and kill chain state (Chapter 6).

Another area of future research is the improved inclusion of the temporal domain. Currently, the order of events is maintained only within process and thread context (see 5.3.4). This allows for an investigation of sequences but does not consider the delay between two behavioral instances. As part of further abstraction efforts it is planned

[1] https://phaidra.fhstp.ac.at

to prepend a temporal identifier to each event, which will transport information about the relative time and duration of execution.

In terms of validation, future work will encompass further proof of soundness for the attribute grammar specification used in the chapter. Furthermore, we will formally test our behavioral engine against evaluation systems such as the one introduced by Filiol et al. [99]. Specific applications like the anomaly detection functionality discussed in Section 5.6.2 will also be evaluated using an even larger and more diverse set of behavioral traces to determine the best suited application scenarios.

On the knowledge discovery side, it is planned to continue development of the KAMAS visualization tool presented in Section 5.6. Specific functionality enabling further statistical assessment will be included to facilitate (malware) forensics, automated sample classification, and various intrusion detection scenarios coupled with a database of explicit domain knowledge.

In general, the automated cross-integration of visual analytics and knowledge discovery methods will be an integral part of our future research into the practical applications of the Sequitur approach.

5.9 Summary

In this chapter we presented SEQUIN, a grammar inference system based on the Sequitur compression algorithm that constructs a context-free grammar (CFG) from string-based input data. The system is capable of automatically assessing arbitrary corpora using statistical means. This enables the accurate identification of both representative (frequent) or anomalous patterns in sequential traces. On the formal side, SEQUIN uses a CFG enhanced with attributes to help describe the extracted (malicious) actions. The inferred rules can be used for flexible unsupervised anomaly detection as well as for compressing AIDIS's input, thereby reducing processing times and building the foundation for future applications beyond the Windows ecosystem. In summary, this chapter contributes by:

- Defining an attribute grammar capable of depicting sequential behavior while retaining information about the triggering process and its parameters;
- Presenting a grammar inference framework based on the Sequitur algorithm capable of performing input data compression and anomaly detection on arbitrary system traces;
- Expanding this approach to a knowledge discovery system supporting automated evaluation and extraction of potentially interesting patterns through our novel KAMAS visualization tool.

We have successfully tested the induction and analysis system with several classes of input data. When used to streamline traces for computationally expensive processes such as AIDIS Core, we achieved a significant reduction in complexity by extracting

representative variables that describe relevant patterns. In our tests, the mean processing time for polynomial graph operations that are the basis of our anomaly detection and classification system was reduced by more than 70% – a vital contribution to the overall solution.

The results presented in Section 5.6 demonstrate the feasibility of using our grammar inference approach over systems that rely on the manual definition of rules. Anomaly detection based on the rule density metric showed promising results by identifying deviating traces and their behavioral sequences in close to real time. Attack detection accuracy of an evaluated ICS sensor data scenario was well above 92% with a specificity of close to 91% (100% each when removing faulty benign runs), while each and every deviating behavioral pattern (benign and malicious both) was successfully extracted for further investigation. With KAMAS, we additionally introduced a visual analytics platform that uses the generated data to assist analysts in locating, extracting, and classifying relevant rules. SEQUIN proved to be a useful supplementation to AIDIS's other components, as it offers the only unsupervised approach to learning malicious behavior, which can be utilized to further dissect anomalies during sentiment mining and star anomaly detection. Future versions of the overall system may also draw on the resulting verdict as additional feature for threat classification.

Overall, the introduced grammar inference tool is most effective when employed to quickly and accurately discover and highlight recurring patterns in sequential sets of arbitrary host and network event traces, thereby aiding in bridging the semantic gap between captured data and attacker behavior. The wide range of tested applications makes SEQUIN a unique tool in the repertoire of malware analysts and researchers alike. Its key contribution to AIDIS is the white-box anomaly induction functionality and the event compression routine able to improve performance across all stages.

Chapter 6

Gamified Attack/Defense Modeling

Contents

6.1 Introduction

Attacks on IT systems are a rising threat against the confidentiality, integrity, and availability of critical information and infrastructures. At the same time, the complex interplay of attack techniques and possible countermeasures makes it difficult to appropriately plan, implement, and evaluate an organization's defense. More often than not, the worlds of technical threats and organizational controls remain disjunct.

Detection and mitigation systems usually offer little in terms of contextual interpretation, which would help to better understand attacker motivations and objectives [187]. At the same time, attack pattern lists, vulnerability databases, and mitigation control catalogs often provide topically constrained information that can rarely be correlated, thanks to varying levels of granularity or abstraction. Most importantly, few models provide a means to map concrete system events as seen on affected assets to a semantic interpretation, known system flaw, or definitive countermeasure. Time and system interdependencies are often not considered. In short, the increasing complexity of cyber-attacks makes it vital to explore novel approaches to attack/defense modeling, threat intelligence, knowledge extraction, and malicious activity detection on multiple layers, while preserving the flexibility needed to encompass new trends and scenarios.

Tailored awareness programs [288, 310] and workable risk assessment procedures [35, 173] have been identified as key components in any successful defense strategy. One approach to combine attack semantics with possible countermeasures and in-depth threat intelligence is the development of so-called 'serious games' [186], a topic that was first broached in the late 17th century as a feasible approach to education in general. Research [78, 122, 261] and security standards/guidelines such as the IT Grundschutz catalog [41] of the German Federal Office for Information Security emphasize that such games are well-suited to teach information security and awareness principles to an audience of differing IT background.

PenQuest, the APT roleplaying game (RPG) introduced in this chapter, is a serious game based on a multi-tiered attacker/defender model that uses gamification, i.e. "game design metaphors to create (a) more game-like and engaging experience" [200]. Players can utilize the RPG to learn in an entertaining way about digital threats and how they may affect different infrastructures. PenQuest also provides an abstraction layer that helps assess the risk of targeted attacks on IT systems by providing an extensible framework for simulating attacker and defender behavior in an adversarial setting. Shortcomings in current defense implementations are inadvertently highlighted during a game session. Managers are encouraged to explore new threats and are presented with possible organizational and system-level countermeasures as suggested by accepted security standards. Analysts and IT/security experts are provided with the means to depict concrete hacks through attack patterns and cyber-observables and link them to the output of intrusion detection systems currently in place at their organization. We use existing repositories and languages for many of PenQuest's inherent concepts to be in line with common industry practices and to enable the future development of

interfaces between our meta model and existing platforms. Specific mappings include the Common Attack Pattern Enumeration and Classification (CAPEC) schema [219] for describing attacks, various actor and asset elements from the Structured Threat Information eXpression (STIX) language [223], exploit properties extracted from the Common Vulnerabilities and Exposures (CVE) database [221], and security controls for (federal) information systems & organizations from NIST Special Publication (SP) 800-53 [146].

Altogether, the gamified model introduced in this chapter helps to raise awareness and supports risk assessment procedures while closing the gap between high-level kill chains, independently published countermeasures and policies, and concrete attacks depicted by real-word events. This versatility turns PenQuest into the solid foundation for an in-depth risk assessment tool as well as framework for simulating attacks based on real-world threat intelligence – in addition to supporting awareness education in higher education and corporate environments.

The remainder of this chapter is structured as follows: After exploring related work in both the intrusion modeling and game (theory) domains (Section 6.2), we discuss the theoretical aspects of our model and present definitions, a high-level view on the multi-layered approach employed, as well as core mechanics (Section 6.3). In Section 6.4, the rule system of the gamified model is presented in detail, ranging from actor definition to assets, countermeasures and vulnerabilities to the actual game phases, as well as offensive and defensive actions. Concrete examples are used to describe the interplay of attack patterns and controls. *Italicized* segments present a cohesive example of applied gameplay across all subsections. Section 6.5 specifies our practical approach to combining STIX vocabularies, CAPEC patterns, CWE and CVE entries, as well as NIST SP 800-53 controls into one model, thereby uniting existing data sources across the threat information domain. Our first physical prototype design is evaluated in Section 6.6. Features and limitations of the model are discussed in Section 6.7. Section 6.8 concludes the article and provides a research outlook. All acronyms used can be found in the Appendix.

6.2 Related Work

Since PenQuest incorporates concepts from attack modeling as well as serious gaming, we take a closer look at similar works in both areas.

6.2.1 Threat Modeling

Several researchers have introduced attacker/defender models that consider numerous factors and properties. In the following, we discuss the most influential works for the development of PenQuest. Like our solution, the Diamond Model of Intrusion Analysis [43] establishes the basic elements of generic intrusion activity, called an event, which is composed of four core features: adversary, infrastructure, capability, and vic-

tim. It extends events with a confidence score that can be used to track the reliability of the data source or a specific event. While some of its premises are similar to our own work, the Diamond Model does not consider Enablers or Disablers (see Section 6.3.2) or any automation for determining specific actions on the attacker's and defender's side. While it is a powerful template in its own regard, PenQuest provides these mechanisms – and more. Its gamified core is synonymous to a ready-to-use framework for simulation and automated knowledge discovery. In summary, the Diamond Model and PenQuest share commonalities and could potentially benefit from each other in terms of feature modeling and terminology.

In the work by Syed et al. [303], the authors present a unified cyber security ontology (UCO) extending the Intrusion Detection System ontology by Undercoffer et al. [317]. UCO is a semantic version of STIX with a link to cyber security standards similar to the ones used in PenQuest. Real-world knowledge is appended using featured Google searches (Google knowledge graph) and various knowledge bases. Syed et al. provide little information about data retrieval mechanisms and general automation. The main use cases emphasized are the identification of similar software and the association of vulnerabilities with certain (classes of) products. Unlike PenQuest, UCO does not consider temporal information or measurements of uncertainty.

Most other works in this area do not attempt to model attack–defense dynamics in a holistic manner while maintaining the link to concrete on-system actions. Existing models usually focus on specific threats or information security aspects [227, 284], formally depict attack–defense trees [161, 267], or describe ontologies with different topical focus [97, 333]. Refer to the survey by Mavroeidis and Bromander [206] and Chapter 2 for additional related work on threat models, ontologies, and languages.

6.2.2 Game Theory and Serious Games

On the side of game-theoretic approaches, models are usually separated into several classes, depending on a multitude of factors. Roy et al. [268] survey a number of approaches for the network security domain and categorize non-cooperative games into static and dynamic scenarios of varying levels of information quality and completeness. As the work provides a good starting point for literature review, we classified PenQuest in accordance to Roy et al.'s schema (see Section 6.3.3).

Cook et al. [63] present an overview of risk assessment strategies for industrial control systems (ICS) that includes, among others, the game theoretic approach. The authors conclude that there exists no unified risk model for hitherto unconsidered ICS scenarios that incorporates events, threats, vulnerabilities, and general consequences with a measure of uncertainty. Suggested future work includes the development of an environment to allow an intelligent adversary to test an ICS's defenses in a nondestructive (e.g. game theoretic) manner. This is specifically addressed by PenQuest. Similarly, Lewis [178] offers an introduction to risk assessment strategies for critical

infrastructure protection, including an in-depth look at Bayesian belief networks and game theory applications.

Addressing a power grid scenario, Holmgren et al. [130] introduce a model for studying defense strategies against hostile actions leading to a power shutdown. Outages are split into stages ranging from prevention to recovery. Unlike the more flexible PenQuest, Holmgren et al.'s model is a perfect and complete information game that does not allow adding new components to the existing network of assets. In addition, only qualified attackers of static skill, determination, and existing access to the system are considered. The overall motivation of implementing the power grid game is still comparable to our approach: It aims to help with resource planning, risk screening, and with studying generic mechanisms that enhance the overall understanding of attacks against a specific system.

Contrary to the above, Nguyen et al. [235] formally describe an abstract zero-sum, incomplete information game for network security scenarios. Their network model is based on the concept of linear influence networks [224], which is represented by two weighted directed graphs that signify the relationship of security assets as well as denoting vulnerability correlation among the nodes. The resulting influence matrix describes the contribution of an asset to overall security. If an asset is taken down by the attacker, its node is removed from the network, lowering the security rating for all connected entities. For common assets, compromising one node changes the probability that another linked asset will come under attack. Unlike PenQuest, Nguyen et al. do not consider specific actors, assets, scenarios, or data sources. Since compromised nodes are removed entirely from the network, the model does not directly support stepped attacks or semi-successful attacker actions. As a pure zero-sum game, it also does not offer the same level of flexible payoff as the system introduced in this chapter.

In the area of serious games, Shostack [290] present Elevation of Privilege, a game designed to motivate people to engage in threat modeling. It is based on the STRIDE mnemonic [160] and includes the Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege attack categories. Defense actions and further threat specifics supported by our RPG are not part of the game.

Another example is Operation Digital Chameleon [260], a red-team exercise turned into a board game. Unlike PenQuest, the game asks of the players to build an attack and defense strategy without the guidance of rules. A game master is responsible for assessing both teams' solutions and encourages discussion. While this offers flexibility and is suitable for dedicated workshops comprising large groups of participants, Operation Digital Chameleon does not provide a model for APT representation and data mapping. With its formal approach, PenQuest paves the way for future scenario computation and automated mitigation planning.

More specialized learning solutions include What.Hack [339], a game revolving around phishing defense, an approach to generate social engineering awareness [27], and a game called OWASP Cornucopia[1], intended for use in software development.

On the side of light entertainment, several vendors have released strategy or card games centered around hacking. Examples include d0x3d[2], Control-Alt-Hack[3], as well as Hackers and Agents[4]. Neither of these games attempt to model overly realistic infrastructures or provide a holistic view on the topic of information security. With PenQuest, we seek to remedy these shortcomings and provide a gamified model that offers realistic scenarios while still being classified as edutainment.

6.3 Attacker/Defender Model

Our gamified approach is based on a novel attacker/defender model tailored to depict interconnected services that maintain – and operate with – various types of information. Consuming and providing services are represented by parent and child entities that enable the modeling of arbitrary, interdependent systems and infrastructures. At the same time, PenQuest incorporates on-system events that define nominal and anomalous behavior, thereby establishing a link to actual attacker behavior in the form of individual actions performed.

Below, we take a closer look at the foundational components of PenQuest: The *base model* defines information and events in the context of the game and outlines how an infrastructure of services can be modeled. The *game model* provides the classification and definition of an "action" as performed by an actor. Lastly, the *rule model* introduces the game-theoretic properties of the RPG.

6.3.1 Base Model

The PenQuest base model consists of three main layers – service, information, and event – that are influenced by attacker and defender actions. Figure 6.1 provides an overview of the interplay of the three main layers and sketches the link to the PenQuest game model. In the following, we present the components that describe the cornerstones of our approach.

Service Layer

In most modeling scenarios, services are synonymous to assets a defender seeks to protect. They will most typically be IT systems maintained by an organization but might also represent a single process running on a computer system.

[1] https://www.owasp.org/index.php/OWASP_Cornucopia
[2] https://www.thegamecrafter.com/games/-d0x3d-
[3] http://www.controlalthack.com/
[4] http://www.hackersandagents.com

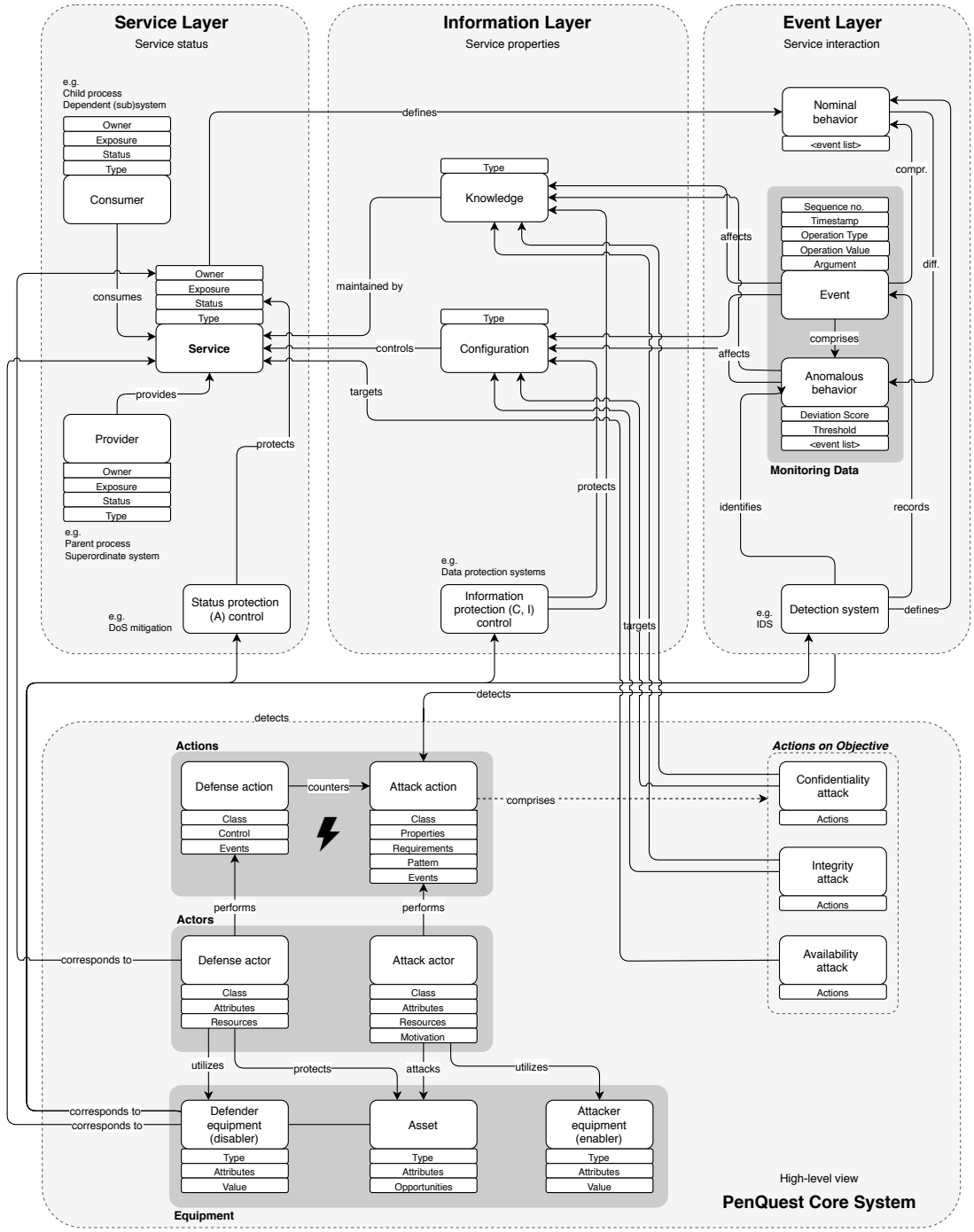


Figure 6.1: High-level view on the PenQuest meta model. The top half represents the layers of the base attacker/defender model while the lower portion depicts the gamified rule system, which is discussed in detail below.

In the model, a service has 4 properties: Type, status, exposure, and ownership. The *type* of a service can be understood as combination of a unique designation and the description of its purpose (functionality). In many cases, the type property will simply contain the name of the service for reference.

Status describes the current level of confidentiality, integrity, and availability of a service. It is the measure of a service's ability to provide its functionality to a consumer, be that another service or human user. In this context, *confidentiality* [299] describes the preservation of secrecy of information associated with the service. If confidentiality is compromised, the respective knowledge or configuration (see Section 6.3.1 below) becomes known to the attacker. *Integrity*, on the other hand, maintains the immutability of information belonging to the service. If integrity is lost, entities can no longer trust in the information being correct. Lastly, *availability* denotes the prevention of system failure or degradation of service functionality. If availability is compromised, a service can no longer be used in an unimpeded fashion.

Exposure determines the placement, and by extension, the exposure of a service to attacks originating outside the infrastructure. Exposed services can be accessed directly, while internal services can only be attacked once an exposed service in the path of attack has been compromised. This principle generally represents the vector used by an adversary to gain access to a specific system.

The *owner* of a service is synonymous to the defending party in our attack/defense model. Owners can be organizations, physical and legal persons, or even abstract entities. In the game rules, the owner is often referred to as Player 2, or Bob.

Services can be reliant on other services for maintaining their functionality. Superordinate services are called *providers*, while *consumers* are in turn dependent on the service in question. The loss of confidentiality, integrity, or availability of the provider may in turn lead to the loss of confidentiality, integrity, or availability of the consumer. More information about service dependencies can be found in the game rules in Section 6.4.3.

Information Layer

PenQuest distinguishes two types of information: Knowledge and configuration. *Knowledge* represents potentially sensitive information relevant to the entity owning and/or using the service. Services are often designed to safeguard knowledge – be that that their main purpose or only implied functionality. Attacks against the confidentiality status of a service seek to access restricted knowledge against the owner's will, while integrity attacks aim to (covertly) change that information.

Configuration denotes the technical or organizational settings and policies controlling a service's functionality as well as its level of exposure. It influences how a service performs its work. Confidentiality attacks against a service's configuration will lead to the disclosure of settings and associated service properties (type, status, expo-

sure). A loss of configuration integrity, on the other hand, might introduce undesired functionality and can change a service's exposure level.

In short, a service is controlled by configuration while maintaining knowledge. Services and information are protected by controls (technical or organizational measures aiming to protect the service from adversary activity) that focus on safeguarding confidentiality, integrity, and availability.

Event Layer

Events are the link between the base attacker/defender model, the PenQuest RPG game/rule system, and actual real-world monitoring data. They formalize activity that leads up to service compromise and describe what an attacking actor is actually doing. Service operation itself is described by events as well, providing a baseline of nominal behavior that can be used for anomaly detection.

Events have a number of properties, including arguments, type, value, temporal information, and an incrementing number in a potential sequence of events. In combination, events can be used to describe a wide range of happenings, starting from undesired operating system (OS) process behavior to more high-level activity influencing an entire (IT) system. For example, a user-land OS process might trigger the event `E(sequence, operation, argument) = (1, create-file, dropped.exe)`, followed by the event `(2, start-process, dropped.exe)`, which results in an IDS anomaly A with a specific value denoting its deviation from a set baseline: (`A(deviation, threshold) = (22.4, 10)`). This occurrence would affect the confidentiality of knowledge that is associated to the modeled service of e.g. `browser.exe`, thereby changing the confidentiality status of the service to 'compromised'. On a less granular IT service level, a generic event `E=(1, change-configuration, htaccess)` caused by the purposeful alteration of a web server's security settings would affect the configuration on an integrity level, changing the service's integrity status to 'compromised' and altering the information controlling the service in the process.

Since we do not only want to sketch such simple attack scenarios but also bridge the gap between actors, assets, vulnerabilities, concrete attack patterns, events, and countermeasures, we extended the model with formally defined *actions* which, in turn, shape the foundation for the PenQuest ruleset.

6.3.2 Game Model

As a model that adopts several concepts from the genres of role playing and strategy games, PenQuest contains a few base building blocks that need to be understood before going into detail: The sides of the attacker/defender contest are represented by *actors*, or characters, that have a certain class. *Classes* describe the background and affiliation of the actor and come with certain properties such as skill, motivation, and monetary resources, which, in turn, are named attributes. *Attributes* are the main determining

factor when it comes to defining an actor's 'strength'. They can change under certain circumstances and are modified through equipment. *Equipment* comprises physical appliances, tools, and more general policies – everything that will provide bonuses or penalties to one side or another. Such *modifiers* typically directly or indirectly influence the chance of success of actions performed in the game. *Actions* are the bread and butter of PenQuest and describe what each actor is actually doing to attack or defend the infrastructure of services that is emulated by the scenario. Actions often come with specific *requirements* in terms of attributes, which reflects the difficulty of the actor's endeavour. The final decision whether an action succeeds usually entails a random factor which is resolved by 'rolling the dice', i.e. computing the outcome according to the modified probability of success.

In the following, we discuss the definitions that are make up above concepts. Depicted in the lower half of Figure 6.1, PenQuest differentiates three main elements: actions, actors, and equipment. The glue between is provided by meta information in the form of categorization, various action requirements, and the annotation of events as well as (third party) attack patterns that are used to populate the model.

Actions

Actions are at the core of the model and tie together all the other components. They also link real-world service and actor behavior to concrete data points such as observable attack patterns or event sequences. Formally, an action X is defined as n-tuple of typical length $n = 11$, whereas the model's flexibility allows for the omission of unneeded elements. Simply put, an action is performed by an actor and is further enabled or disabled by various types of equipment. It is assigned a category within the model, which includes usage requirements, properties pertaining to its success, and a detection chance. Attack patterns and associated events tie it to real-world data points.

$$
\begin{aligned}
X = \langle & \\
& \langle AttackActor\langle Class, Motivation, Attributes, Resources\rangle\rangle, \\
& \langle DefenseActor\langle Class, Attributes, Resources\rangle\rangle, \\
& \langle Enabler\langle Type, Effect, Attributes, Name\rangle\rangle, \\
& \langle Disabler\langle Type, Effect, Attributes, Name\rangle\rangle, \\
& \langle Victim\langle Type, Name, Exposure, Parent, VectorParent, Configuration, \\
& Knowledge, Status, Integrity\rangle\rangle \\
& \langle AttackClass\langle Stage, PatternClass, Mode\rangle\rangle, \\
& \langle DefenseClass\langle Category, ControlClass, ActionClass\rangle\rangle, \\
& \langle Requirements\langle *Actor\langle Attributes, Resources\rangle, Victim\langle Exposure, Integrity\rangle\rangle, \\
& \langle Properties\langle Sophistication, SuccessChance, DetectionChance\rangle\rangle, \\
& \langle AttackPattern\langle Impact, ID\rangle\rangle, \\
& \langle Event\langle Type, Time, ScoreSequence, Parent, Operation, Argument\rangle\rangle\rangle
\end{aligned}
$$

Actors

Actors are the players of PenQuest. The defending actor (service owner) tries to fend off an attacker while his or her adversary seeks to compromise the status of a targeted service. Our model differentiates two actor classes: The *AttackActor* and the *DefenseActor*, which correspond to the actor types introduced in the `ThreatActorType` vocabulary schema of the STIX threat information language [23]. Each attacker is described as a unique class with their own motivation, primary attributes in the form of Sophistication (measure of an actor's skill, SO), Determination (measure of actor motivation and the strength of their cause, DE) and Wealth (financial resources of an actor, WE), as well as operational resources such as Initiative (INI) and Insight (INS): Initiative (i.e. time efficiency) is an attribute derived from SO and DE that determines the number of overall actions each actor can perform within a given period of time. Generally, each action in the strategy set has a time requirement, making this attribute the main resource for all attack and defense activities. Insight, on the other hand, measures the cumulative knowledge gained about the opponent and thereby increases the overall chance of success when attacking/defending.

The $\langle AttackActor \rangle$ classes currently modeled in the RPG include: Cyber Espionage Operations (TH), Hacker (white (EX), gray (RO), black hat (RA)), Hacktivist (CR), State Actor/Agency (OP), Insider (IN), and Disgruntled Customer (PR). See Section 6.4.1 for more detailed information.

Each attacker class has a range of possible motivations. These goals represent the overall objective of the attack and provide additional context for determining attacker attributes and skills. They correspond to the STIX Threat Actor `Motivation` vocabulary and encompass various ideological goals ($id.*$), as well as ego-centered (eg), financial (fi), military (mi), opportunistic (op), and political (po) motivations. The actor creation routine of PenQuest supports the automation of the entire process and provides percentages defining likely actor/goal combinations.

$$
\begin{aligned}
AttackActor = \langle & \\
& \langle Class\{TH, EX, RO, RA, CR, OP, IN, PR\}\rangle, \\
& \langle Motivation\{id.*, eg, fi, mi, op, po\}\rangle, \\
& \langle Attributes\langle SO\{1..n\}, DE\{1..n\}, WE\{1..n\}\rangle\rangle, \\
& \langle Resources\langle INI\{1..n\}, INS\{1..n\}, \langle Enabler\rangle\rangle\rangle\rangle
\end{aligned}
$$

Following the same formula, $\langle DefenseActor \rangle$ classes include the three primary sectors (CP, CM, CS), infrastructure (IF), military (MI), state actors/agencies (SA), the education sector (ED), and private individuals (PI).

$$
\begin{aligned}
DefenseActor = \langle & \\
& \langle Class\{CP, CM, CS, IF, MI, SA, ED, PI\}\rangle, \\
& \langle Attributes\langle SO\{1..n\}, DE\{1..n\}, WE\{1..n\}\rangle, \\
& \langle Resources\langle INI\{1..n\}, INS\{1..n\}, \langle Disabler\rangle\rangle\rangle\rangle
\end{aligned}
$$

Equipment

There are two types of equipment in PenQuest. So-called *Enablers* represent attack tools and vulnerabilities employed by the aggressor. They have specific effects on the target service and come with a wide range of properties (e.g. level of impact or maturity of a vulnerability-type Enabler (see CVE/CVSS mapping in Section 6.5)) and prerequisites (e.g. privileges and level of user interaction required) that need to be considered. Attacker equipment ($ATT.*$) subsumes mostly attack tools (MPT, Pwd), OS, application, and (wireless) network scanners (Sca), various types of malware (Mal), as well as vulnerabilities (VUL) that modify the chance of success against specific assets.

$$
\begin{aligned}
&Enabler = \langle, \\
&\quad \langle Type\{ATT.*, VUL.*\}\rangle, \\
&\quad \langle Effect\langle Type\{incSC, decDC\}, EffectValue\{1..n\}, \\
&\quad EffectTarget\{H, N, I, M, T\}\rangle\rangle, \\
&\quad \langle Attributes\langle Sophistication\{1..n\}, Level\{1, 2\}\rangle, \\
&\quad \langle Properties\langle Privileges\{high, low\}, \\
&\quad UserInteraction\{none, required\}, \\
&\quad \langle Impact\langle C\{low, med, high\}, \\
&\quad I\{low, med, high\}, A\{low, med, high\}\rangle, \\
&\quad Maturity\{unproven, PoC, functional, high\}\rangle\rangle\rangle, \\
&\quad \langle Name\{\_ToolName\_\}\rangle \\
&\quad \rangle
\end{aligned}
$$

Defenders use *Disablers* to thwart their adversary. Such defender equipment encompasses assets to be protected (AST), security policies (POL), fixes (FIX) that counter vulnerabilities, as well as tools that increase security ($SEC.*$), e.g. through prevention (Pre), detection (Det), delay (Del), or by generally hindering the attacker (Cnt). The respective effects aim to reverse or mitigate the damage caused by attacker equipment and actions. Specific examples can be found in the game rules in Section 6.4.2.

$$
\begin{aligned}
&Disabler = \langle, \\
&\quad \langle Type\{AST.*, SEC.*, POL.*, FIX.*\}\rangle, \\
&\quad \langle Effect\langle Type\{decSC, decSC\}, EffectValue\{1..n\}, \\
&\quad EffectTarget\{H, N, I, M, T\}\rangle\rangle, \\
&\quad \langle Attributes\langle Sophistication\{1..n\}, Level\{1, 2\}\rangle\rangle, \\
&\quad \langle Properties\langle Maturity\{official, temporary, \\
&\quad workaround\}\rangle\rangle, \\
&\quad \langle Name\{\_SystemName\_\}\rangle\rangle
\end{aligned}
$$

Some enablers and disablers only target or apply to specific equipment categories. The general $\langle Effect\rangle$ of using a piece of equipment depends on its type and is exemplified on a per-item basis in the game rules. Typically, various resources, attributes,

and detection/success ($*DC/*SC$) probabilities are either increased ($inc*$) or decreased ($dec*$) upon use, which further impacts future events and the overall course of the game. For the $\langle EffectTarget \rangle$, we generally differentiate host-based (H), network-based (N), industrial (I), mobile (M), and third-party (T) equipment. Enablers generally have an adverse $\langle Impact \rangle$ on the CIA status of the victim (more below).

The *Victim* class describes the system that is targeted by the attacker, typically a specific service (i.e. asset). Victim information differentiates between internal and exposed systems – which can be accessed directly from within the DMZ – and identifies service status, integrity, and systemic as well as attack vector parents that represent consumers, providers, and systems that have to be compromised before the victim can be attacked directly.

In each gamified scenario, the attacker attempts to achieve his or her goal by compromising a victim (target) in a specific fashion. This translates to the attempted compromise of one the defender's assets by changing its $\langle Victim \langle Integrity \rangle \rangle$ to 'compromised' or its $\langle Victim \langle Status \rangle \rangle$) to 'stopped'. There are three kinds of possible attacks on any asset, in accordance to the CIA triangle of information security [299]: Confidentiality attacks (theft of information), integrity attacks (altering of information), and availability attacks (service status changes), which are all part of the $\langle AttackClass \rangle$ meta component introduced below.

$$
\begin{aligned}
Victim = \langle & \\
& \langle Type\{Asset, SecuritySolution\}\rangle, \\
& \langle Name\{\_SystemName\_\}\rangle, \\
& \langle Exposure\{internal, exposed\}\rangle, \\
& \langle Parent\{\langle Victim\rangle, \langle Disabler\rangle\}\rangle, \\
& \langle VectorParent\{\langle Victim\rangle\}\rangle, \\
& \langle Configuration\{tech, org\}\rangle, \\
& \langle Knowledge\{Information, \langle Configuration\rangle\}\rangle, \\
& \langle Status\langle OperationStatus\{running, affected, stopped\}\rangle, \\
& \langle Integrity\{nominal, affected, compromised\}\rangle\rangle\rangle
\end{aligned}
$$

Meta Information

Each action is assigned an *AttackClass* by the model. Specifically, this class contains the categorization into the APT kill chain stage [133, 188] ($\langle Stage \rangle$), and discriminates various types of attack pattern. These types correspond to the mapping of granular actions to CAPEC attack patterns [219], which is discussed in detail in Section 6.4.5. $\langle Mode \rangle$ identifies the optional CIA goal [299] and impact rating of the action, which, if successful, is reflected in the victim service's $\langle Status \rangle$ information. *AttackClass* information helps to abstract specific attack actions and provides a means to link adversary behavior to possible defense measures.

$$
\begin{aligned}
&AttackClass = \langle \\
&\quad \langle Stage\{R.*, W.*, D.*, E.*, I.*, C.*, A.*\}\rangle, \\
&\quad \langle PatternClass\{IG, IN, SE, SA, FA, BF, IA, DM, PR, CO\}\rangle, \\
&\quad \langle Mode\langle Goal\{C, I, A\}, Rating\{low, med, high\}\rangle\rangle\rangle
\end{aligned}
$$

The corresponding *DefenseClass* contains our abstraction of NIST SP 800-53 controls [146], which we use to link attack patterns to specific technical and organizational countermeasures. See Section 6.5 for more information.

$$
\begin{aligned}
&DefenseClass = \langle \\
&\quad \langle Category\{organization, information\_system\}\rangle \\
&\quad \langle ControlClass\{IL, CP, AW, SP, FI, AP, AC, DI, SI, CS\}\rangle \\
&\quad \langle ActionClass\{ACM, ACE, IFE, LEP, REA, AMO, RST, \\
&\quad COM, CCC, COS, COP, COP, AUM, INH, NOM, MES, \\
&\quad PAC, SEP, SCP, BOP, CRP, FLR, MCP, ISM\}\rangle\rangle
\end{aligned}
$$

Requirements identify the minimum actor attributes and resources needed to conduct the attack, as well as other prerequisites in the form of system exposure. Requirements are optional and depend on the complexity of the respective attack, which is defined through its *Properties*, namely minimum required actor skill (Sophistication SO) and the chance of success (SC) as well as detection (DC). This information is largely gleaned from CAPEC and CVE.

$$
\begin{aligned}
&Requirements = \langle \\
&\quad \langle *Actor\langle Attr.\langle SO\{1..n\}, DE\{0..n\}, WE\{0..n\}\rangle\rangle\rangle, \\
&\quad \langle *Actor\langle Resources\langle INI\{1..n\}, INS\{1..n\}\rangle\rangle\rangle, \\
&\quad \langle Victim\langle Exposure\{internal, exposed\}\rangle\rangle, \\
&\quad \langle Victim\langle Integrity\{nominal, affected, compromised\}\rangle\rangle\rangle
\end{aligned}
$$

$$
\begin{aligned}
&Properties = \langle \\
&\quad \langle Sophistication\{1..n\}\rangle, \langle SC\{0..100\%\}\rangle, \langle DC\{0..100\%\}\rangle\rangle
\end{aligned}
$$

Each action corresponds to an observable *AttackPattern*, which is identified by its ID and its impact on the CIA triad. Attack patterns link the modeled action to specific hostile activity as described in the CAPEC schema.

$$
\begin{aligned}
&AttackPattern = \langle \\
&\quad \langle Purpose \langle Exploitation\{T, F\}, Obfuscation\{T, F\}, \\
&\quad Penetration\{T, F\}, Recon\{T, F\} \rangle \rangle, \\
&\quad \langle Impact \langle C\{low, medium, high\}, I\{low, medium, high\}, \\
&\quad A\{low, medium, high\} \rangle \rangle, \\
&\quad \langle ID\{n\} \rangle \rangle
\end{aligned}
$$

Ultimately, attack patterns are mapped to a set of monitored *Events* with specific underlying operations and arguments, triggers (parents), timestamps, anomaly scores, and sequence numbers. The 'pattern' type describes event sequences that directly represent the action, while anomalies describe the behavioral deviation from a baseline. The modeling of unique events closes the gap to the data layer and allows us to lower the level of abstraction to the actual systemic representation. Refer to Section 6.5 for a mapping example.

$$
\begin{aligned}
&Event = \langle \\
&\quad \langle Type\{Pattern, Anomaly\} \rangle \\
&\quad \langle Time \langle Start\{\_ timestamp\_\}, End\{\_ timestamp\_\} \rangle \rangle, \\
&\quad \langle Score \langle Deviation\{0..n\}, Threshold\{0..n\} \rangle, \\
&\quad \langle Sequence\{n\} \rangle, \\
&\quad \langle Parent\{\_ ParentName\_\} \rangle, \\
&\quad \langle Operation\{\_ OperationName\_\} \rangle, \\
&\quad \langle Argument\{\_ OperationArgument\_\} \rangle \rangle
\end{aligned}
$$

6.3.3 Rule Model

The definitions discussed above provide the distinct elements that are part of the model and introduce relationships and concrete class definitions. Before these components can be assembled into a playable game, we need to outline the game-theoretic principles of PenQuest as well as its core mechanics.

Game Principles

Instead of relying solely on decision models, adversarial behavior can be expressed in the form of game situations [178]. Techniques associated with game theory, which is defined as *"the study of models of conflict and cooperation between rational decision-makers"* [266], can be used to model multi-player scenarios that allow participants to pursue goals and rewards by acting in the most favorable, risk-adverse, or cost-effective way possible. The outcome of a game presents the actors with a strategy for resource allocation, risk minimization, or a general approach to meeting a certain objective.

Even though PenQuest is a model/game hybrid, it can be categorized according to several principles of game theory, as summarized by Roy et al. [268]:

- **Non-cooperative nonzero-sum game**: Opposition between players is an integral part of the design: Player 1 (attacker) always combats Player 2 (defender) and tries to achieve adverse goals by stealing information, manipulating the integrity of data or systems, or by shutting them down entirely. Even though actions are not typically assigned points that are symmetrically gained/lost (making it nonzero-sum), it can be argued that the mechanism of asset compromise is in fact a zero-sum game, where the defender loses points describing integrity and status, while the attacker gains a corresponding advantage. In other situations, win/loss is represented by an increase or decrease of attributes or action success and detection chance. These bonuses are one-sided, yet always shift the balance between the players away from equilibrium.
- **Asymmetric strategy**: The strategy sets of the two players are not identical – the attacker draws from a different pool of actions than the defender. This stems from the difference in goal and purpose: Attackers will attempt to penetrate a system using malware or by exploiting vulnerabilities, while a defender tries to counter these actions by implementing technical and organizational controls.
- **Dynamic/extensive game with static elements** [268]: While the game uses sequential moves characteristic for dynamic games, the second player typically remains unaware of the first player's actions, making the model bear some resemblance to a strategic setting where players act simultaneously and in secret. At its core, PenQuest remains dynamic – emphasized by its multi-stage nature.
- **Imperfect, incomplete information**: As stated above, Player 1 does not necessarily know the moves previously made by the attacker, and vice versa. It is in fact vital to players' success that performed actions remain secret, thereby potentially causing the other party to make imperfect decisions. At the same time, the general set of strategies is known to both sides. The exact payoff in a certain situation, however, is not, due to the lack of information about past activity and their impact on success and detection chances (incomplete information).
- **Bayesian formulation of static elements**: In PenQuest, players have incomplete information on the other players, especially when it comes to actions and strategies, which are derived from the attacker's type and ultimate objective. There is, however, a fixed probability that players, being one of n available classes, need to conduct/defend against one of three kinds of attacks on a finite set of assets in order to win the game.
- **Finite & discrete**: While some action combinations are continuous in nature, the general action/reaction game follows a discontinuous sequence. The number of game turns is limited by an exhaustible resource – Initiative (i.e. time efficiency).

Following Roy et al.'s taxonomy [268], our game can be classified as non-cooperative, dynamic game with imperfect and incomplete information, that draws

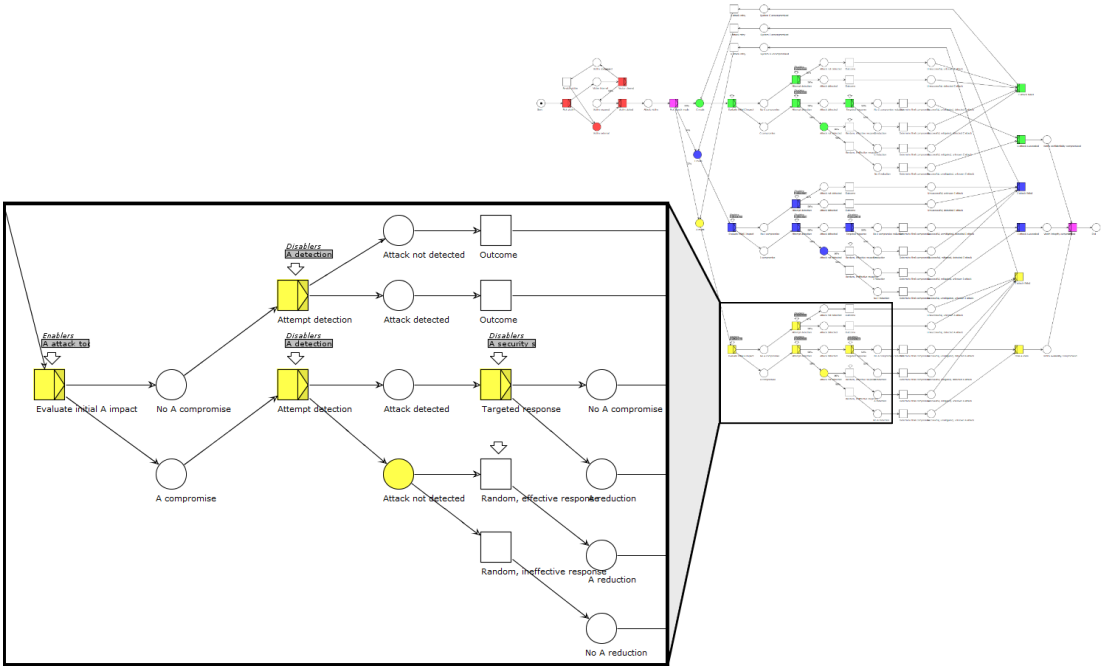


Figure 6.2: WF-net describing the process of picking and attacking a victim. The net consists of 64 places, 51 transitions, and a total of 133 arcs. Soundness was determined by analyzing the net in WoPeD [107].

from static elements of Bayesian formulation. According to You and Shiyong [351], our two-player hybrid zero-sum game G is defined by the triplet:

$G = (A, D, F)$, where:

- A is a set of strategies of Player 1 (attacker),
- D is a set of Player 2 (defender) strategies, and
- L is, for select game principles (see below), a real-valued function $A \times D$. Therefore, $L(a, d)$ is a real number for every $a \in A$ and $d \in D$.

Core Mechanics

The core mechanics of the game include the model's sole zero-sum component describing target compromise and a full attack–response mechanism for achieving hostile or defensive goals. The central part of the model is synonymous to the process of picking a $\langle Victim \rangle$ service and executing an attack of a certain $\langle AttackClass \rangle$ corresponding to the CIA triangle mentioned above, followed by a matching defensive response. We have modeled this basic building block as a Workflow net (see Figure 6.2) to identify inconsistencies in the model. A workflow net (WF-net) is a strongly connected Petri net (PN) with two unique input (source) and output (sink) places as well as a reset transition r.

Following the notation of Esparza et al. [93], a Petri net can be defined as a 5-tuple $N = (P, T, F, W, m0)$ where P is a set of places or states, T is a set of transitions with $P \cap T = \varnothing$, $F \subseteq (P \times T) \cup (T \times P)$ is a flow relation, $W : (P \times T) \cup (T \times P) \to N$ is a

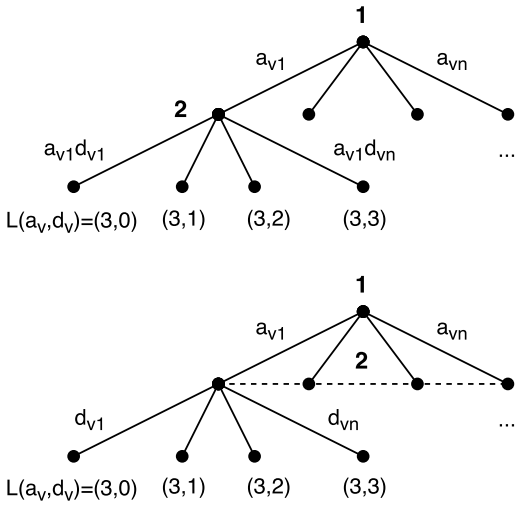


Figure 6.3: Information set of the defending player for victim system compromise through attacker action a_{v1}, shown in extensive form [168]. In the above case, Player 2 successfully unveils a_{v1}, giving him the chance to specifically counteract the gain $L(a_v)$ of Player 1 by increasing his own score $L(d_v)$ through defense action d_{v1}. Below tree represents the limited information set for Player 2, when a_{v1} remains undetected.

weight function satisfying $W(x, y) > 0 \iff (x, y) \in F$, and $m0 : P \to N$ is a mapping called the initial marking. In our case, $m0$ equals the initial state where a victim has not yet been picked (called "Start"). A reset transition r leads back to the beginning of the attack process and repeats until the victim has been successfully compromised.

Specifically, an attacker picks a $\langle Victim \rangle$ that is either exposed or in-vector (see Section 6.3.1). Subsequently, a mode of attack (C, I, or A) is chosen. As depicted in Figure 6.2 the initial impact of such an attack is determined based on the $\langle Enabler \rangle$ resource and various other properties and attributes. Independently from the level of success of the action, the defender first attempts to detect the activity. In the Petri net, this is again modeled as exhaustible resource that corresponds to the $\langle Disabler \rangle$ class, where detection systems (Det) can be found. If an attack was detected, the defending entity can attempt to counter the initial impact by utilizing further $\langle Disablers \rangle$. Ultimately, initial impact and the degree of success in reducing that impact is resolved in PenQuest's only zero-sum game element:

Here, Player 1 chooses action $a \in A$ targeted at victim $v \in V$ ($\langle Victim \rangle$ in our class structure), which is kept secret unless Player 2 meets the detection requirements (see Figure 6.3). If action a_v is conducted successfully, Player 1 gains a number of points $L(a_v)$ representing the level of victim compromise, which are directly or indirectly deducted from the respective defender's tally (payoff). Independently from the outcome of the detection attempt, Player 2 chooses $d \in D$ for victim $v \in V$, attempting to counter the (assumed) action a_v. Points $L(d_v)$ are computed, fully negating $L(a_v)$ in the best of cases.

This game principle applies to victim system integrity and status (see Figure 6.3 as well as the $\langle Victim \rangle$ definition in Section 6.3.2), the success chance of hostile attacks

and defensive actions (see $\langle Properties \rangle$ and $\langle Enablers \rangle / \langle Disablers \rangle$ below), and some other attribute and resource bonuses and penalties discussed in the next chapter.

We have again modeled this principle as WF-net, describing the process of generating $L(a_v)$ as well as $L(d_v)$, respectively. This second net, with a total of 33 places and 65 transitions, can be used for simulating arbitrary actions generating $L = 0..3$ on both the attacking and defending side. The unveiling of actions is again part of the parent WF-net discussed above.

It is important to bear in mind that the RPG's victory conditions are only indirectly affected by these shifts in L, and that an increased numeric distance from the equilibrium of factors other than victim system integrity and status does not always guarantee one player's domination over the other. In fact, victory is determined by the exhaustion of available (temporal) resources or the successful compromise of the chosen victim asset within an allotted time window.

In the following, we build upon these mechanics and construct a full-fledged strategy game for simulating real-world attacks and their possible countermeasures.

6.4 Game Rules

In order to transform the model into a playable format while exemplifying both education and data enrichment purposes, we opted to cast the rules of PenQuest in the mold of a game manual. This approach is also owed to the fact that our system adheres to the principle rules of a roleplaying or board strategy game for two players that follows a set of stages to determine the course of a play-through. Conditional decisions are made by rolling dice, which is representative for outcomes that come with a certain chance of success as well as a random element.

In the following, we present the three main aspects of PenQuest, which build the foundation for our physical prototype. In the preliminary stage, dubbed character or *actor creation*, the players specify the two opponents, their basic attributes and skills, as well as attack/defense goals corresponding to specific objectives and motivations. Once the actors are defined, the game is ready to start. Specific activities within the boundaries of this system are represented by *conflict actions* that, combined into a full play-through, constitute an attack targeting a service asset. See below for detailed information on each of these core game rules.

6.4.1 Actor Creation

Characters are the main actors involved into an attack/defense scenario. Since PenQuest is a very flexible game system, there are many possible combinations of opponents and scenarios. Instead of offering a fixed set of adversaries, we provide a full-fledged character creation system that supports both manual and automated actor design. Generally, it is important that the player portraying the attacker keeps all their properties secret

– neither class, goal, attributes, nor the available equipment should be known to the defender. In contrast, the class of the defender is public knowledge. Defender attributes and equipment are initially kept secret, but may be uncovered during the reconnaissance phase (see Section 6.4.4). See Figure 6.4 for a full overview of the character creation process.

Class and Motivation

The respective player starts by picking in secret a *class* for the attacking side ($\langle AttackActor \rangle$). Classes follow the `ThreatActorType` vocabulary schema of STIX [23]. Third party eCrime actors such as money laundering services and malware developers are not modeled, since we want to focus on active factions that are likely to directly target the assets of a victim. The available classes (STIX name) are:

1. **Thief *TH*** (Cyber Espionage Operations): Cyber spies are entities aiming to steal information such as intellectual property or other private data (knowledge).
2. **Explorer *EX*** (Hacker, White hat): This class of hackers does not seek to cause damage but acts to improve current security measures by e.g. raising awareness.
3. **Rogue *RO*** (Hacker, Gray hat): Grey hat hackers sometimes violates regulations but usually lacks the malicious intent typical for a black hat hacker.
4. **Raider *RA*** (Hacker, Black hat): Black hats are malicious cyber-actors that do not operate within legal or moral boundaries.
5. **Crusader *CR*** (Hacktivist): These hackers assume the moral high ground and typically want to prove a point or leave an ideological statement.
6. **Operative *OP*** (State Actor/Agency): State actors can be agencies with various missions in the areas of (clandestine) intelligence or counter-intelligence.
7. **Infiltrator *IN*** (Insider threat): Insider threats are individuals of varying skill that seek to undermine their own employing organization.
8. **Protester *PR*** (Disgruntled customer/user): This class of actors represent a generally unsatisfied third party.

Each character class has a range of possible *motivations*. These represent the overall objective of the attack and provide additional context for determining attacker attributes and skills. Motivations correspond to the Threat Actor `Motivation` vocabulary that is part of STIX as well as the *Motivation* class of our attacker/defender model. Possible motivations $\langle AttackActor \langle Motivation \rangle \rangle$ are:

- Ideological (*id*): The actor acts out of their ideological belief in a cause, such as anti-corruption, anti-establishment, environmental, ethnic/nationalist, information freedom, religious, security awareness, or human rights.
- Ego (*eg*): The attacker wants to prove a point to others or herself.
- Financial or Economic (*fi*): The attack is motivated by financial goals.

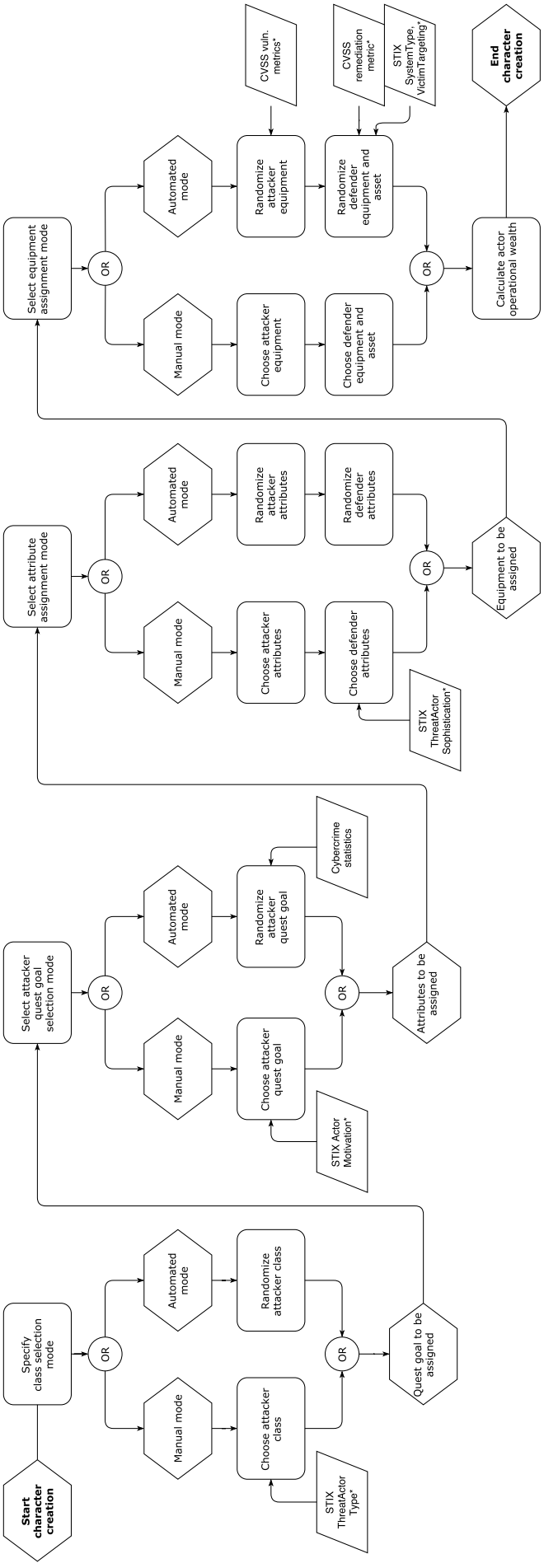


Figure 6.4: Actor creation process overview. Data imports marked with an asterisk (*) are used in both the respective manual and automated mode of actor creation.

- Military (*mi*): The actors wants to achieve a military victory or gain an strategic/tactical advantage.
- Opportunistic (*op*): The attacker ceases an unexpected opportunity to strike against a target.
- Political (*po*): The adversary acts out of political motivation.

While a player can pick any motivation from the above list when manually creating a character, not every attacker is equally likely to have a certain motive. Disgruntled customers will not typically have political agendas, while black hat hackers are unlikely to be driven by an environmentalist ideology. The likelihood of combinations is modeled by the RPG rule system as percentage values denoting the likelihood of a character/-motive mapping. Randomization is achieved by rolling two ten-sided dice (2D10) and then consulting the probability Table 6.1, whereas the first D10 determines the tens digit and the second D10 represents the unit position. Currently, these motivations are largely cosmetic and help to flesh out the actors. However, it is possible to expand the rule system by incorporating cyber-crime statistics that reflect typical motivations.

Armed with attacker class and motivation, we can specify the defending actor (defender class, $\langle DefenseActor \rangle$) of the hostile campaign by simply choosing from below list or by rolling another D8. The list of actors is inspired by STIX' `Victim Targeting by Sector`, which leverages the external CIQ (Customer Information Quality) standard published by OASIS[1]. However, there is no exhaustive `Industry Type` list provided by STIX. We therefore compiled our own list of target actor types:

1. **Company, Primary Sector *CP***: Private company operating in the primary sector (e.g. mining, farming, forestry).
2. **Company, Manufacturing *CM***: Company in the business of non-infrastructure manufacturing and processing.
3. **Company, Services *CS***: Company in the services sector.
4. **Infrastructure *IF***: Organization maintaining (critical) infrastructure.
5. **Military *MI***: (Para-)military organization with strategic or tactical focus.
6. **State Actor/Agency *SA***: State-sponsored organization with possible intelligence background.
7. **Education *ED***: Schools, universities and other organizations in the education sector.
8. **Private individual *PI***: Person without relevant external affiliation.

In the following and throughout this chapter, we use an example scenario with the actors of Alice and Bob to explain PenQuest's rules. Alice personifies the attacker and Bob represents the defending actor.

[1] https://stixproject.github.io/documentation/idioms/industry-sector/

	TH	EX	RO	RA	CR	OP	IN	PR
Ideological	0-10	0-25	0-16	0-10	0-40	0-10	0-15	0-30
Ego	11-29	24-45	17-32	11-20	41-50	11-15	16-35	31-50
Financial	30-65	46-55	33-49	21-50	51-55	16-40	36-55	51-70
Military	66-70	56-60	50-66	51-65	56-60	41-65	56-60	71-75
Opportunistic	71-90	61-80	67-83	66-80	61-70	66-75	61-90	76-90
Political	91-100	81-100	84-100	81-100	71-100	76-100	91-100	91-100

Table 6.1: Typical motivation of the various attacker classes. The numbers represent the lower and upper bound of the range. Ideological sub-goals are currently not covered, but may be easily added for additional granularity.

	TH	EX	RO	RA	CR	OP	IN	PR	CP	CM	CS	IF	MI	SA	ED	PI
SO		+1				+1	−1			+1		+1		+1		
DE				+1	+2		+1	+1			−1	+1				+1
WE	+1	−1			−2	+1	−1		+1	+1	+2	+1			−1	−2
INI	−1		+1		+1	−1		+1		−1		−1	+1			+2
INS	+1	+1					+2	−1				−1			+2	

Table 6.2: Default attribute modifier table for all actors. For custom gameplay or simulation scenarios with a reduced need for balancing, these values can be freely changed and should be based on the latest available studies.

> *Alice decides to use automated character creation to put together an APT scenario. She rolls an eight-sided die (D8) to determine attacker class and scores a '5', making her offensive actor a Crusader (hacktivist). When it comes to motivation, she rolls '6' and '5' on a D10. Consulting the probability table, she determines the attacker motivation to be Opportunistic, meaning that the Crusader simply chanced upon the occasion. Next, Alice rolls a '7' for the defender class. Her chance victim (Bob) seems to be in the education sector.*

Attributes

PenQuest uses *attributes* to distinguish different levels of actor skill, motivation, and resources. These can be set manually for a custom game or determined randomly. Attributes have ratings that range from 1 to 5, whereas 1 corresponds to an undeveloped state and 5 to the highest possible maturation level. Attributes in PenQuest are *Sophistication*, *Determination*, and *Wealth*:

Sophistication *SO* – Measures an actor's capabilities, determines Initiative (number of actions during conflict, see Section 6.4.1 below), the base success chance of actions (see Section 6.4.5), and limits the use of more sophisticated equipment. *SO* is each actors' primary attribute, as defined by the STIX threat actor vocabulary `Threat-ActorSophistication`. We have opted to use the following levels of Sophistication: Aspirants (1) show little to no technical capabilities and are usually accidental perpetrators (attacker) or common users (defender). Novices (2) have basic computer skills. Practitioners (3) are versed in using automated tools. Defending practitioners have a

workable knowledge of the best practices and utilize pre-configured security solutions. Experts (4) have in-depth knowledge about system internals and operational security. Innovators (5) are masters of their field and create their own tailored tools and solutions.

Determination *DE* – Denotes the actor's overall motivation and drive. It influences Initiative and, in limited deck mode (see Section 6.4), the number of actions available to the players. For attackers, Determination specifies how intent they are to reach their goal, while a defender's motivation simply quantifies the typical level of effort invested into protecting a given asset. We define five levels of Determination: Indifferent (1) actors are not particularly interested in achieving their attack or defense goal – their willingness to invest time or resources is next to nil. Casual (2) determination represents a "for fun" attitude and might be representative of an intrinsically and extrinsically barely motivated individual [203]. Willing (3) actors will invest an average amount of resources into their mission without sacrificing other duties. Attackers or defenders classified as Devoted (4) will disregard work hours or will dedicate significant resources to their task. Finally, Zealous (5) actors will do everything in their power to achieve their goal, including breaking laws or disregarding personal health.

Wealth *WE* – Specifies the actor's available monetary resources that can be spent on equipment (see below). Wealth is typically dependent on the respective attacker and defender class. Certain organizations are more likely to have financial constraints for cyber operations or information security than others, as is apparent in Table 6.2. Like any other attribute, Wealth comes in 5 levels: Destitute (1) actors do not have any noteworthy budget, the Provident (2) rating is used for the financial assets of a typical middle-class individual, Endowed (3) describes the average operational budget of a small company, Rich (4) actors command funds of a successful medium-sized business, and Prosperous (5) organizations/individuals can easily spend corporate-size budgets on a campaign or defense objective. Specific numbers may be attached to these ratings, if desired. For simplicity's sake, each point of Wealth translates to 5 fictional credits that can be spent on $\langle Enablers \rangle$ and $\langle Disablers \rangle$ during character creation and setup.

Certain classes may come with modifiers to their attributes, representing an increased or decreased likelihood of having a distinctively high attribute score. These modifiers are listed in Table 6.2.

> *Continuing our above example, Alice now needs to specify the attributes for her Opportunistic Crusader and the victim education organization. She rolls a D6 six times, re-rolling eventual sixes. She scores '3', '2', and '4' for the attacker. As the Crusader class has modifiers as listed in the Table 6.2 (+2 Determination, −2 Wealth), the final attribute levels are Sophistication (3), Determination (4), and Wealth (2). On the defender side, there are additional modifiers. Bob's rolls of '4', '2', and '4' are translated into the final values of '4', '2', and '3' (-1 Wealth).*

Resource Pools

Next to the actors' main attributes, there are two additional actor resources that come into play when pitting two characters against each other. The first is *Initiative* (*INI*) or *time efficiency*. Initiative is a derived attribute that determines the number of overall actions each character can perform. Each of the actions introduced in Section 6.4.5 has an Initiative requirement and also uses a certain number of Initiative points when performed, making this attribute the game's main resource for attack and defense activities. There are actions that increase or decrease Initiative. This is representative for the time required by a hostile action and for how certain (counter)measures might influence the amount of time an actor has left to successfully thwart the attack. The game ends in a victory for the defender when the attacking actor has exhausted their Initiative without compromising the target asset.

The second derived resource is *Insight* (*INS*). Both attack and defense actions benefit from knowledge about the opponent. The more one party knows about the other, the more likely his or her chance of successfully conducting an action. The initial Insight Pool is zero, but might be modified by certain class properties. In the course of the game, the Insight Pool will gradually increase for both actors – enabling new actions and giving them the chance to better react to their adversary. The base detection chance of the defender is linked to the action category of the attack (see 6.4.5).

> *Alice and Bob determine Initiative for their two sides: With a Sophistication and Determination rating of 3, Alice's Crusader has an Initiative score* $(SO*2)+DE$ *of* 9*. The education institution targeted is rated at Initiative* $(4*2)+2=10$*, which is boosted to 11 by the planning policy that Bob implemented during setup. The unmodified Insight Pool for the attacker is 0 (no class bonuses), while the defender receives a +2 class bonus.*

6.4.2 Equipment

Equipment in the game serves multiple purposes. It represents the target of a hostile action, the systems used to support or thwart such an attack, and general tools and policies that can be bought and implemented. Equipment always comes at a monetary cost measured in credits and derived from the Wealth attribute of the actor (see Section 6.4.1). Tables 6.3 and 6.4 provide an overview for the respective equipment types. While it is still possible to purchase equipment during later stages of the game (see Section 6.4.4), a baseline of an actor's technical capabilities has to be established right off the bat. Both attacker and defender procure equipment individually without informing the opponent.

Attacker equipment mostly comprises attack tools ($\langle Enablers \rangle$) as per our base model) that modify the chance of success against specific target assets or actors by increasing their operator's attributes or base success chance values. Other tools (hack-

ing suites, vulnerabilities) come with a Sophistication rating of their own, which can substitute for the attacker's attribute for operations not directly executed by the actor.

Defender equipment ($\langle Disablers \rangle$) encompasses tools that increase security, detect, or mislead the attacker. The effectiveness of a piece of equipment is measured by its contribution to detection or evasion as well as its action success chance modification ($\pm$ percentage points).

In general, equipment is divided into the following categories, denoted in the game model as $\langle\langle\langle Attack\ Actor \rangle Enablers \rangle Type \rangle$ and $\langle\langle\langle DefenseActor \rangle Disablers \rangle Type \rangle$.

Disablers

The following presents the types of equipment available to the defending actor, which includes security solutions, policies, and the systemic assets themselves.

Assets (AST) are determined by the defender's class and are awarded automatically at no cost. It is the goal of the attacker to compromise one of these assets in order to win the game. See Section 6.4.3 for more information and Table 6.5 for character—asset allocation.

Security solutions (SEC) – Security solutions are technical systems and tools intended to prevent, detect, delay, or generally hinder hostile actions. In the context of the game, they are physically or logically connected to an asset such as the network or a user workstation, making it harder to successfully attack the service and its consumers. Specifically, we differentiate the following systems (also see Table 6.3):

- Prevention solutions (Pre) – increase the difficulty of successfully attacking an asset by assigning a penalty to the action's success chance ($decSC$). See Section 6.4.5 for more information on action success.
- Detection measures (Det) – boost the defender's ability to identify hostile actions by directly increasing the defender's detection chance ($incDC$).
- Delay solutions (Del) – increase the amount of time and effort required by the attacker to perform the hostile action. In game terms, delay systems increase the Initiative cost of the respective attack ($incINI$).
- Recovery solutions (Rec) – decrease the impact of hostile attacks after the fact. Specifically, recovery solutions can reduce the level of compromise of C, I, or A attacks ($decIMP.*$).
- Countermeasures (Cnt) – describe generic technical solutions that improve the defender's ability to counter hostile attacks. Countermeasures provide a flat boost to the defending actor's Sophistication attribute in specific scenarios ($incSO.*$).

Security policies (POL) – Security policies are non-technical measures that increase security on an organization level. Similarly to security solutions, they increase defender attributes, resources, or grant special abilities. However, the bonus applies to the de-

⟨*Type*⟩	⟨*Name*⟩	⟨*Effect*⟩	⟨*EV*⟩	⟨*ET*⟩	Cost
Pre	Host-based IPS I	*decSC*	5	H (all)	1
Pre	Network-based IPS II	*decSC*	10	N (all)	2
Pre	Content-based IPS I	*decSC(D.D)*	15	H (all)	0.5
Pre	Rate-based IPS II	*decSC(A.A)*	30	N (all)	1
Pre	Web application firewall I	*decSC(R.S,D.I)*	15	Web	0.5
Det	Host-based IDS II	*incDC*	10	H (all)	2
Det	Network behavior analysis system I	*incINS*	1	N	0.5
Det	Log analysis system II	*incDC*	10	H	1
Del	Sandbox I	*incINI*	1	H	0.5
Rec	Failover network II	*decIMP(A)*	2	N	2
Cnt	Stateful firewall I	*incSO*	1	all	2

Table 6.3: Exemplary list of security solutions (*SEC*). Level II systems always cost the double amount of credits to implement but also offer twice the bonus (⟨$EffectValue(EV)$⟩). Systems with ⟨$EffectTarget(ET)$⟩ = * (all) apply their benefit to all disablers of that type, while others can only be attached to one asset or security solution.

fending organization as a whole, making them a powerful tool for establishing a secure baseline. Policies cost time and funds to implement. Below list is taken directly from the NIST SP 800-53 standard ([146], see Section 6.5 for more information about mappings and Section 6.4.5 for more details about NIST-based defense actions) and cover the following aspects (NIST control ID):

- Access control (AC-1): Combines access control systems, policies, and session control mechanisms.
- Awareness & training (AT-1): Increases defender Sophistication by implementing organization-wide awareness and training measures.
- Audit & accountability (AU-1): Relates to audit-based non-repudiation measures.
- Security assessment & authorization (CA-1): Evaluation of controls and their adherence to e.g. external standards.
- Configuration management (CM-1): Encompasses establishing and maintaining performance and functionality of systems throughout their life cycle.
- Contingency planning (CP-1): Describes contingency plans such as fail-overs to alternate storage or processing sites.
- Identification & authentication (IA-1): Manages session control and (cryptographic) identification and authentication measures.
- Incident response (IR-1): Encompasses procedures, handling, monitoring, and reporting of incidents as well as their follow-up response.
- Maintenance (MA-1): General maintenance of systems, such as update policies and downtime schedules, are part of this policy.
- Media protection (MP-1): Subsumes access, designation, storage, transportation, sanitization, use, and downgrading of physical media containing relevant data.
- Physical & environmental protection (PE-1): Physical access control and environmental protection enforcement.

⟨*Type*⟩	⟨*Name*⟩	⟨*Effect*⟩	⟨*EV*⟩	⟨*ET*⟩	SO	Cost
MPT	Host exploit kit I	*incSC*	5	H (all)	1	1
MPT	Pentesting software II	*incSO*	2	all	2	4
Sca	Mobile OS scanner I	*decDC*	5	M (all)	-	0.5
Sca	ICS analysis tool II	*decDC*	10	I (all)	-	1
VSc	Cloud vuln. scanner I	*reduces VUL.cpx*	low	T (all)	-	1
NSc	Network mapper	*incINS*	1	N (all)	-	1
Pwd	Password cracker II	*incSC(D.I)*	10	H,N,I,M,T	-	1
Mal	Rootkit I	*decDC*	5	H,N,I,M,T	1	0.5
Mal	Ransomware II	*incCR*	4	H	2	1
Mal	Spambot I	*decINI(D.D)*	1	H	1	0.5

Table 6.4: Exemplary list of attack tools. Level II systems always cost the double amount of credits to implement but also offer twice the bonus ($\langle EffectValue(EV)\rangle$). Systems with $\langle EffectTarget(ET)\rangle$ = * (all) apply their benefit to all disablers of that type, while others can only be used to attack one asset or security solution.

- Planning (PL-1): Meta-policy for the design of information security architecture, secure operation, and central management.
- Personnel security (PS-1): Manages personnel screening, transfer, tracking, and termination.
- Risk assessment (RA-1): Policy for the risk assessment and vulnerability scanning process.
- System & services acquisition (SA-1): Revolves around the system development life cycle, the allocation of resources, as well as the procurement of (third-party) systems and services.
- System & communications protection (SC-1): Manages defense measures related to DDoS protection, shared network resources, crypto policies, VoIP, wireless link protection, I/O device assess and usage restrictions as well as numerous other factors contributing to secure (inter-)system communication.
- System & information integrity (SI-1): Includes countermeasures to malware, system monitoring, alerts, function validations, error handling, and other, primarily remediation-centered information protection activities.

Enablers

Listed below is the equipment available to the attacker:

Attack tool (ATT) – Attack tools are malicious programs coded to perform reconnaissance (scanners) circumvent security (hacking tools) or automate certain offensive tasks. They either provide a flat bonus or need to be attached to attack actions as one-time modifiers. See Table 6.4 for an exemplary list. Attack tools are categorized into:

- Multi purpose tool (MPT): These hacking tool-sets allow the attacker to automatically probe and exploit known vulnerabilities without purchasing a specific attack. If the attacking actor operates such a tool, they can decide to use the

tool's Sophistication attribute for related actions instead of their own. The specific capabilities (actions that can be automated) differ from tool to tool and range from an increase in success chance ($incSC$) to an increase in Insight ($incINS$) or Sophistication ($incSO$).

- System scanner (Sca): Specific to each class of equipment (see Table 6.4), these tools increase the knowledge about a system ($incINS$) or aid via a reduction of the detection chance ($decDC$), thereby enabling more complex attacks. There are scanners for every type of asset that need to be coded or procured individually for each $\langle EffectTarget \rangle$ category.
- Vulnerability scanner (VSc): Vulnerability scanners determine the existence of weaknesses in host-based systems. In game terms, they reduce the complexity requirements of vulnerability-based attacks performed by the attacker. See Section 6.4.2 for more information on vulnerabilities and exploits.
- Network scanners (NSc): Tools in this category (packet sniffers, port scanners) intercept network traffic and thereby grant insight into the network environment and its connected systems ($incINS$). They additionally expose security solutions installed within the network context.
- Password cracker (Pwd): Primarily used to bypass account security, password crackers either brute-force passwords or attempt to login using a prepared list of likely secrets. Using them increases the success chance ($incSC.D.I$) of Intrusion type ($D.I$) attacks. See Section 6.4.5 for more information about attack phases.
- Malware (Mal): Malware summarizes all software that mirrors the harmful intent of an attacker in an automated fashion. In PenQuest, malware is 'attached' to a successful Delivery action ($D.*$). We differentiate Rootkits (decrease detection chance of a subsequent attack, $decDC$), Backdoors (increase chance of success and decrease detection risk ($incSC$, $decDC$)), Ransomware (generate credits ($incCR$)), Trojans (similar to multi purpose tools and some scanners), as well as Botnet Zombies (reduce Initiative costs for certain kill chain phases ($decINI.req$). Like some multi purpose tools, malware has its own Sophistication level for determining its success. With the exception of backdoors, all malware can only be used once and expires after it has been triggered.

Exploits and Fixes

Exploits and fixes have a special role in PenQuest. While also equipment by definition, exploits provide the means to lower the e.g. Sophistication requirements of an attack by aiding the attacker in her efforts. Fixes available to the defender directly counter specific exploits. The concept is detailed in the following:

Exploit (VUL) – A (technical) vulnerability is a flaw in a specific asset or security solution that makes it easier for the attacker to breach the system by exploiting it. They require a certain level of Sophistication to use and might demand additional

privileges. Vulnerabilities generally exist in all types of defender equipment, but are hard to exploit without knowing of their existence. Depending on complexity *cpx*, they need a successful run of a vulnerability scanner on the victim resource (see above) before they can be used. Vulnerabilities are key to compromising the victim asset and therefore for deciding the outcome of the game.

Vulnerabilities as modeled by the RPG follow the metrics defined by CVSS: `Attack Complexity` *cpx* (high or low) determines the actor Sophistication requirements, `Privileges Required` *prv* (true or false) additionally decide the need for pre-existing user privileges, `User Interaction` *usr* (none or required) define whether the vulnerability can exist stand-alone without an accompanying action, and `CIA Impact` *imp.*$*$ represents bonuses (high (+2), low (+1), or none (+0) for each triad factor) that determine the modifier to the accompanying attack's effect. Each vulnerability is additionally rated in accordance with its `Exploit Code Maturity` *mat*, which determines a one-time Sophistication *SO* bonus and monetary cost of the exploit. Vulnerabilities without user interaction are assigned an *SO* value of their own, as hinted at by the Temporal Metric Group of CVSS. This directly affects the success chance of exploiting the respective vulnerability. In short, vulnerabilities also come with a Sophistication attribute or modifier that is derived from their CVSS score, which is linked in turn to the CAPEC attack pattern via their entry in the Common Weakness Enumeration (CWE) database [222]. See Section 6.5 for more information about model mappings.

In PenQuest, vulnerabilities are generally used together with an attack action for additional supporting or enabling effects. Corresponding exploits can either be coded by the attacker (*W.C* action, as discussed in Section 6.4.5), or purchased (see above). Either way, a successful Reconnaissance (*R.*$*$) or *W.P* action is needed to enable the use of vulnerabilities. In our current iteration of the game, we have implemented a number of real-world vulnerabilities including some well-known ones such as ShellShock[1], GHOST[2], and Heartbleed[3].

Fix (*FIX*) – Fixes directly counter vulnerabilities or address other weaknesses in existing assets. They come in several classes as per the `Remediation` metric of CVSS: Official Fix, Temporary Fix, and Workaround. Official fixes (high SO, restore integrity from 'compromised' to 'nominal' (3 increments, *incINT*)) and workarounds (low SO, restore from 'compromised' to 'affected' or 'highly affected' to 'nominal' (1 increment)) are effective indefinitely but are not able to counter zero-day exploits. Temporary fixes (medium SO, restore by 2 increments) are only effective for one game turn and might come with side effects. Workarounds generally come with a lower chance of success, conditional on their Sophistication. See Section 6.4.3 for more information about compromise levels.

[1] https://nvd.nist.gov/vuln/detail/CVE-2014-6271
[2] https://nvd.nist.gov/vuln/detail/CVE-2015-0235
[3] https://nvd.nist.gov/vuln/detail/CVE-2014-0160

> *In our example, the attacking Crusader's Wealth rating is 2. Translating to 10 credits' worth of funds ($Wealth * 5$), this gives the attacker only a few options to prepare for her campaign. Alice spends the immediately available 50% of the money on a host exploit kit (1 credit), a level II vulnerability scanner for database systems (1 credit), and a 'functional' vulnerability for database systems for 2 credits. For the defender, Bob uses 7 of his 15 credits to implement an access control policy (2.5 credits), a planning policy (0.5 credit), a level II host-based intrusion detection system (2 credits), and a level I stateful firewall (2 credits). During the game, the remaining credit amount can be spent on additional equipment. Therefore, Alice's Crusader has an operational budget of 6 credits, while the education organization can spend 8 credits during play.*

6.4.3 Assets and Topology

Before we select the victim of the campaign, we have to take a look at the types of assets available in the game. Assets are determined by the defender's class and are awarded automatically at no cost (see Table 6.5). It is the goal of the attacker to compromise one or several of these assets in order to win the game. Assets are part of the defender's infrastructure and operate within the context of a network (which is an asset itself). Below, we take a closer look at specific assets and some of the more general equipment concepts in regards to the modeled topology.

We generally differentiate the following asset types: Host-based (H), network-based (N), industrial (I), mobile (M), and third-party (T) assets. Model-wise, we use a simplified version of the `SystemType` vocabulary of STIX' `VictimTargetingType` TPP schema to model individual assets (denoted in brackets):

- Application server (*App*, internal, H) (Enterprise Systems–Application Layer): Generic server running an organization-relevant application.
- Database server (DB, internal, H) (Enterprise Systems–Database Layer): Generic data and/or configuration store.
- Network (Net, internal and exposed, N) (Enterprise Systems–Network Systems, Enterprise Systems–Networking Devices): Underlying network connecting all other assets. We differentiate an exposed demilitarized zone (DMZ), a local area network (LAN), and an industrial network (subsumed under the term 'SCADA'). A dedicated internal network is optional for the PI defender class.
- Web server (Web, exposed, H) (Enterprise Systems–Web Layer): Server hosting the public web presence of an organization or individual.
- Communication system (Com, internal and exposed, H) (Enterprise Systems–VoIP, Enterprise Systems–Web Layer): Communications infrastructure including, but not limited to, telephony, e-mail, and instant messaging.

	CP	CM	CS	IF	MI	SA	ED	PI
Application server (*App**)	✓	o	✓	o	o	✓	✓	❉
Database server (*DB**)	✓	✓	✓	✓	✓	✓	✓	
Network segment (*Net*)	o	o	o	o	o	o	o	o
Web server (*Web*)	❉	✓	o			✓	o	✓
Communication system (*Com*)	o	o	o	o	o	o	o	o
Communication system (*Com**)		✓	✓	✓	✓	✓	✓	
Industrial control system (*ICS**)	✓	✓		✓	❉			
Industrial safety system (*ISS**)	❉	❉		✓	❉			
Mobile system (*Mob*)			✓	❉	✓	✓	❉	o
Third party service (*3Pa*)			✓			❉	✓	✓
Workstation (*WS**)	o	o	o	o	o	o	o	o

Table 6.5: Mapping of defense actor classes to controlled assets. o...no Sophistication requirement (all actors of this type control at least these assets), ✓...Sophistication 2-3, ❉...Sophistication 4-5. Asterisk (*): Internal system (LAN or industrial network).

- Industrial control system (ICS, internal, I) (Equipment Under Control, Operations Management, Supervisory Control): System controlling industrial equipment such as manufacturing plants.
- Industrial safety system (ISS, internal, I) (Industrial Control Systems–Safety, Protection and Local Control). Safety systems for prevention and mitigation of disadvantageous scenarios affecting human health.
- Mobile system (Mob, exposed, M) (Mobile Operating Systems, Near Field Communications, Mobile Devices): Mobile devices and (individual) short-range communications tools.
- Third-Party service ($3Pa$, exposed, T) (Application Stores, Cloud Services, Security Vendors, Social Media, Software Update): Services such as cloud storage, outsourced web services, and supplier systems.
- User workstation (WS, exposed for actor PI, otherwise internal, H) (Application And Software, Workstation, Removable Media): Physical or virtual machine operated by the end-user.

When it comes to targeting assets, there are three key concepts that define how a system can be attacked: asset compromise, attack vector, and asset dependency.

Asset Compromise

Asset compromise is the key principle that governs how an attack on a system is modeled in PenQuest. As stated in Sections 6.3.1 and formalized in Section 6.3.3, there are three kinds of attacks with a $\langle Mode\langle Rating\rangle\rangle$ ranging from 'low' (1), 'medium' (2), to 'high' (3). The available mode and its rating is dependent on the action being used; not all actions provide the same level of systemic impact (see Section 6.4.5 for more information). There are three modes of attack of which the attacker can choose one at the time of the hack:

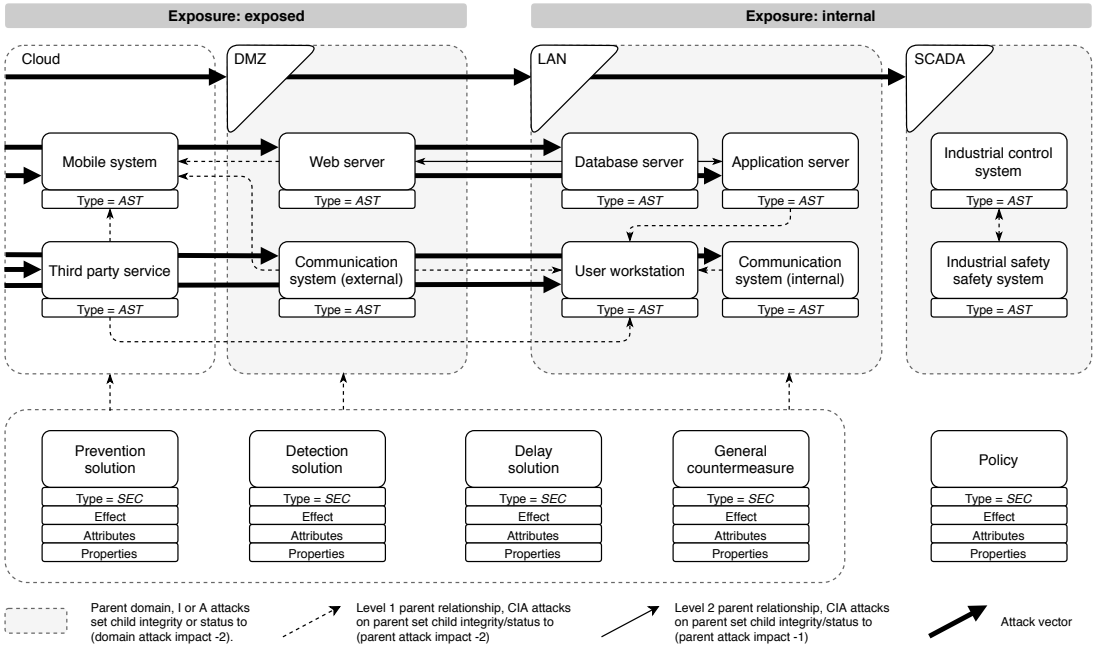


Figure 6.5: Attack vectors and parent–child relationships for both victim exposure levels in the abstracted topology. Security solutions are parents to assets; attacking them on the integrity level can set child integrity to 'affected'. The effect of CIA attacks on assets and other disablers is further detailed in Section 6.4.5.

- **Confidentiality** attacks with a rating of 'high' (3) increase the attacker's Insight pool (*incINS*), but have no further effect on system integrity or status. The effect is cumulative over time: Three successful 'low'-rated (1) attacks or one 'medium' (2) plus one 'low'-rated attack accumulate to the same effect. A successful 'high' (3) level confidentiality attack is necessary to win a game with a data theft (confidentiality) scenario.
- **Integrity** attacks of 'low' (1) and 'medium' (2) rating set the targeted service's integrity (*decINT*) to 'affected' or 'highly affected', respectively. An attack rated 'high' (3) will change integrity to 'compromised', which is required to progress along the attack vector and to win sabotage scenarios. The effect is again cumulative.
- **Availability** attacks target the victim's status ($\langle Victim \langle Status \rangle \rangle$): One or several successful attacks (again dependent on the rating) set the target's status to 'stopped' ($\langle Enabler \langle Effect \rangle \rangle = decSTA$), representing a system that is no longer operational.

Any successful Integrity compromise (rating 3) of an asset or Disabler within the 'exposed' or 'internal' exposure domain enables the attacker to target connected 'internal' systems, which would otherwise be inaccessible (see Figure 6.5). See the next section for more information about the attack vector.

Attack Vector

We model network topology by differentiating between *exposed* and *internal* systems (see Section 6.3.1). In order to attack an internal system, the attacker has to first compromise an exposed parent asset or security solution or proceed along a predefined *attack vector*: The attack vector describes the path an attacker must take in order to reach the desired victim, meaning that any attack on a still uncompromised topology has to target the mobile system (Mob, type M), third party service ($3Pa/T$), external comm system (Com/H), web server (Web/H), user workstation (WS/H), or the exposed demilitarized zone network (Net/N) itself. Incidentally, these are also the systems the defender should do his or her best to wall up. Once a system has been fully compromised integrity-wise (see Section 6.4.5), the attacker can target any system connected to the one he or she just hacked. Figure 6.5 depicts the concept as bold black arrow.

Asset Dependency

Assets and security solutions may be *parent* (provider) to another system. If a parent is successfully attacked, the effect of the compromise is passed on to all children (consumers) at a reduced rate. Specifically, a 'high' (3) level attack on a parent will also have 'medium' (2) impact on the child for level 2 relationships, and 'low' (1) impact for level 1 relationships. An overview of attack vectors and parent–child relationships is depicted in Figure 6.5.

> *The assets that Alice might be after are provided automatically. In the case of our education organization with a Sophistication attribute of 4, Bob the defender controls an application server, a database server, a web server, a communication system, third party services, user workstations, an industrial control system for educational purposes, and various network resources (see Table 6.5).*

Victim Selection

Before the game starts in earnest, the attacker picks the actual *Target* ($\langle Victim\langle Type = \{Asset\}\rangle\rangle$) in the attacker/defender model) and *Mode* ($\langle AttackClass\langle Mode\rangle\rangle$), i.e. one of the defender's assets to compromise by changing its $\langle Victim\langle Integrity\rangle\rangle$ to 'compromised' or its $\langle Victim\langle Status\rangle\rangle$) to 'stopped' (also see Figure 6.5). There are three kinds of possible attacks on any asset: Confidentiality attacks (theft of information), integrity attacks (altering of information), and availability attacks (system status changes). See Section 6.4.5 for more information about attack actions.

For custom games, it is perfectly feasible to specify more varied or numerous goals. An attacker could e.g. attempt to first steal information off an asset before shutting it down. Other scenarios include penetration tests of all assets within a network segment or a DoS attack on two systems simultaneously.

> *Alice decides that the game will end once she successfully alters the User Database asset of the defender (integrity attack of the 'System manipulation' (Action on Objective–Integrity attack) category, changing some of the grades in the process.*

6.4.4 Game Phases

At this point we are ready to play. We differentiate two distinct game modes: In *limited deck* mode, attacker and defender randomly select actions (i.e. a card) from the pool. After an action was used it is discarded from play. At the end of a turn, players draw a new card for each action spent. Limited deck mode has been designed with gameplay mechanics and balancing in mind and is intended to be used in casual games.

In *simulation mode*, players may freely choose from the entire action pool at any given time. Attack actions are only discarded if the defender successfully spotted and identified the hostile activity. This mode represents a more realistic approach where attackers and defenders always have the full strategy set at their disposal.

As depicted in Figure 6.6, the game starts off in *Stealth phase*. This means that the defender is unaware of the looming threat and has to rely on his initial equipment to keep his assets safe. Stealth phase lasts until the defender manages to successfully detect an attacker action. Actions can be performed for free during Stealth phase, meaning that they do not deduct from the Initiative total. Purchasing equipment is done normally at the end of the round. To reflect the reduced activity of the defender during this stage, he or she generates a certain amount of credits every two rounds, providing the actor with additional procurement options. Each round, attacker and defender may spend one Initiative point worth of actions. Typically, the attacker uses this phase to conduct reconnaissance to increase his or her Insight Pool. Defenders typically spend a part of their operational budget to improve security systems, conduct spot checks (use defense actions on systems deemed likely victims) or simply hope for their detection systems to pick up the attacker's activity. If the defender manages to detect a hostile action, the action and its resulting level of compromise of the targeted system is unveiled and Stealth phase ends with the current turn.

> *Alice wants to stealthily assess the target system by conducting a reconnaissance 'Scan' action (see Section 6.4.5). The rounded down success chance of this action is 60% (Alice's base Sophistication rating +3), meaning that she has to score 6 or less on a ten-sided die. Alice rolls her D10 and scores a '5' – well within range of the success threshold of 1-6. As a result, her Insight pool is increased by 1, boosting the base success chance by 5% and enabling a greater range of invasive actions for her subsequent turns. Bob now rolls another D10 to determine if the action was caught by his monitoring systems. With his base detection chance (30%) and an Insight pool of 2 (+10% chance of detecting a hostile action) he needs to score a '4' or less to spot Alice's scan. Equipment on both sides would additionally modify this target value.*

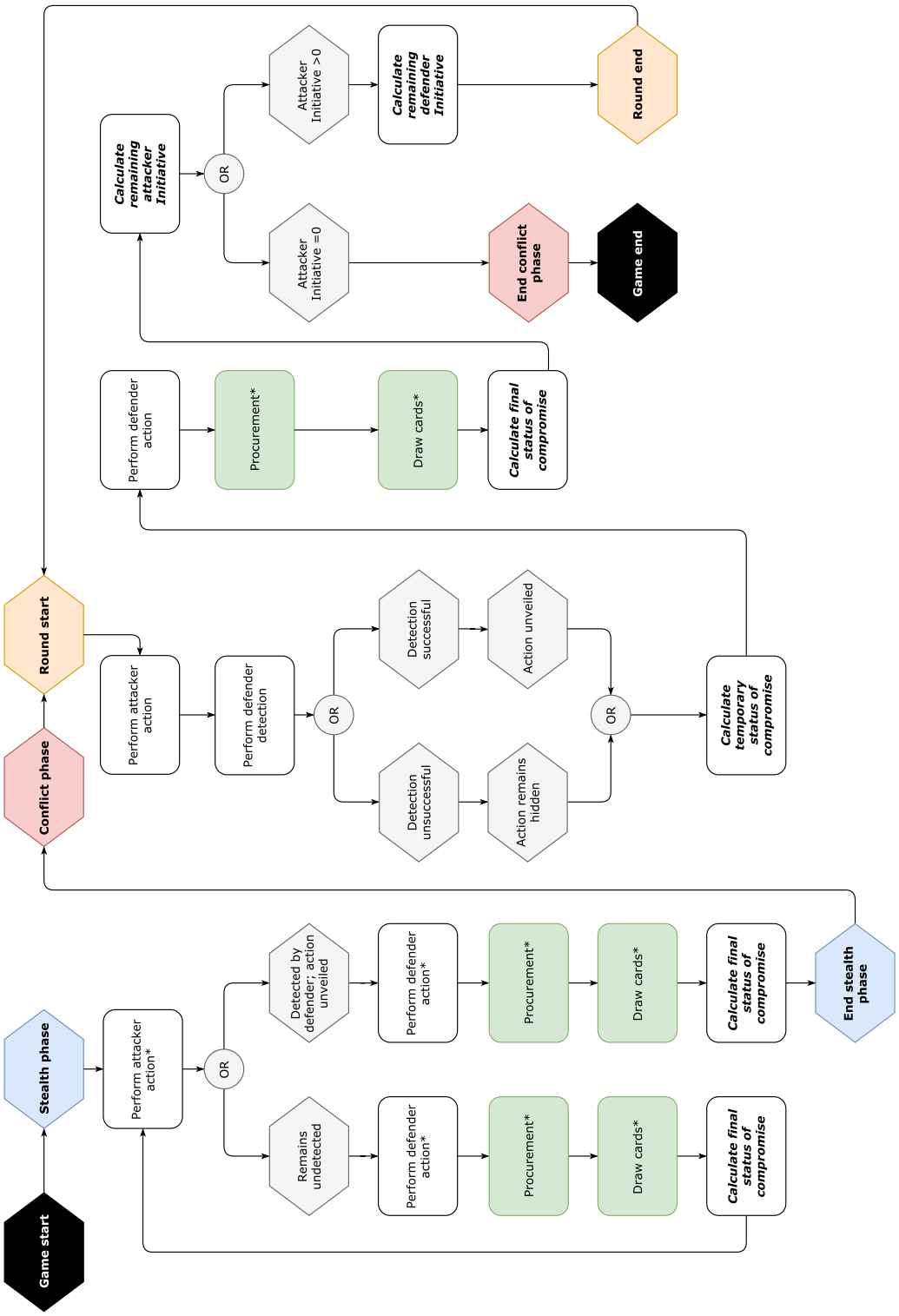


Figure 6.6: Game phases and round progression overview. Actions marked with an asterisk (*) do not reduce the respective actor's Initiative pool. Initiative and equipment status of compromise is recalculated after both actors have completed their actions.

With the end of the Stealth phase the game enters *Conflict phase*. Initiative is now deducted normally for each action performed, which represents the ticking clock. There are seven possible action types that correspond to the APT cyber kill chain by Hutchins et al. [133]. Every APT stage is further split into subcategories [188] that are ultimately linked to specific attack patterns. See Section 6.4.5 and Section 6.5 for more information about the interplay and mapping of offensive and defensive actions.

Attacker and defender continue to act alternately. Each attacker action comes with a success and detection chance (for specifics, see Section 6.4.5) that is modified by equipment on both sides. Whenever the defending actor succeeds in detecting a hostile action, the action itself and the current level of compromise of the system are unveiled, giving the victim the chance to understand the attacker's goals and react accordingly with a defensive action, which, similarly, has a certain chance of being successful.

In addition to triggering a normal attack or defense action, the actors can use their remaining credits to procure equipment from a randomly drawn pool of equipment cards (limited deck mode), or from the full range of enablers/disablers (simulation mode). Equipment is available for use at the beginning of the next turn to model necessary implementation efforts.

Each hostile action further increases the chance of successfully compromising the target asset, while each reaction of the defender will make it harder for the adversary to penetrate the system. Misdirection and timing are key, just like in the real world: It is vital for the attacker to conceal his or her ultimate goal for as long as possible and to strike when the chance of success is greatest. All the while the defender has to protect their assets to the best of their abilities until all Initiative has been used up and the attack has been averted.

6.4.5 Actions

In this section, we discuss PenQuest's action categories as well as the individual actions used to compromise or defend an asset. The core mechanics introduced in Section 6.3.3 are hereby specified and linked to accepted vocabularies and standards.

Actions are at the heart of PenQuest. They represent the concrete attack or countermeasure being used at a given point in time. Each action comes with a success chance and a detection chance modified by actor attributes and equipment currently in play. The attacker first rolls a die (typically a D10 or D100 to represent percentages) to determine whether the action is successful. If the result of the roll is equal or lower than the success chance threshold, the action succeeds. The defender now rolls another die in an attempt to detect the event after the fact. If that succeeds, the current action of the attacker is unveiled.

In the following, we introduce concrete attack and defense actions, which are based on the Common Attack Pattern Enumeration and Classification (CAPEC) dictionary and classification taxonomy by MITRE [219] as well as on the mitigation controls listed

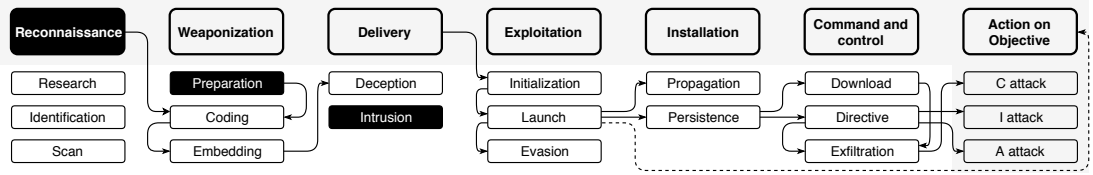


Figure 6.7: APT stage completion dependencies. The arrows represent which stage (i.e. action associated to the stage) needs to be completed before another stage action can be executed ('followed by'). The dashed arrow shows the simplified dependency for quick (non-persistent) attacks. Black boxes represent possible start points for kill chain traversal.

in NIST Special Publication (SP) 800-53 [146]. Attack actions are linked to the APT kill chain by Hutchins et al. [133], which we expanded with several subcategories modeled in the TAON ontology [188]. Figure 6.8 in Section 6.5 summarize the link between kill chain, attack mechanisms, and mitigating controls.

Action Categories

Action categories encompass and classify the individual attack operations conducted by the aggressor. Each subcategory of these kill chain actions has a base detection and success chance ranging from 0 to 100%. The Sophistication attribute of the actor or tool, employed vulnerabilities, special attacks, and various countermeasures might modify that base value. The percentage is currently defined manually by a group of IT security experts, but may easily be altered to match newly released information in future iterations. Some kill chain stages require the successful completion of an attack belonging to another stage prior to execution (see Figure 6.7), depending on desired game complexity: In *APT mode*, dependencies include persistence and the establishment of a C2 channel. *Quick mode* omits these stages and only requires a successful 'Launch' action ($I.L$).

Each specific attack action (see Section 6.4.5) is assigned one or several action categories. Specifically, these APT kill chain categories are:

- **Reconnaissance** ($R.*$): Research into the target and scanning of related assets for information. Subcategories include Research ($R.R$) using public search engines, Identification ($R.I$) of systems through e.g. fingerprinting, and Scan ($R.S$), where a victim system is actively scanned for weaknesses and topological properties. Successful reconnaissance enables the procurement of vulnerabilities.
- **Weaponization** ($W.*$): Preparing exploits and weaponizing code. Weaponization mostly takes place at the attacker's premises and is therefore nigh impossible to detect. Its subcategories are Preparation ($W.P$), which includes exploit searches and targeted research, the Coding ($W.C$) of exploits and tools, as well as Embedding ($W.E$) the prepared or purchased malware in websites, mail messages, or other, ostensibly harmless media.

⟨*ID*⟩*	Name*	Methods*	⟨*PatternClass*⟩	⟨*Stage*⟩	Kn.*	Purp.*	C*	I*	A*
100	Overflow Buffers	Analysis; Injection	IG; IN	R.I; R.S; E.I; E.L	1–3	Pen.; Expl.	3	3	3
103	Clickjacking	Spoofing; Social Eng.	IA; SE	D.I; I.P	3	Expl.	3	3	1
104	Cross Zone Scripting	Analysis; Injection	IG; IN	R.I; R.S; E.I; E.L	2	Expl.	3	3	3
105	HTTP Request Splitting	Proto. Man.; Analysis; Injection	DM; IG; IN	D.I, E.*; R.I; R.S	2	Expl.	2	2	1

Table 6.6: Excerpt from the current pool of PenQuest attack actions. Columns marked with asterisk (*) identify information mined from CAPEC attack patterns. 'Kn.' specifies the attacker knowledge required to perform the attack, ranging from low (1) to high (3). The C/I/A columns denote the respective impact of a successful use.

- **Delivery** (*D.**): Delivery actions describe the process of gaining access to or smuggling payload into the victim's perimeter. Specifically, we differentiate Deception (*D.D*) attacks that use logical or physical social engineering to fool the victim, and straightforward Intrusion (*D.I*): Here, the attacker actively tries to penetrate the target's system using technical means.
- **Exploitation** (*E.**): In this stage, a payload or attack code is actively executed on the system. During Initialization (*E.I*), malware or an exploit is prepared for launch by abusing a system weakness. Launch (*E.L*) describes worker processes, threads, services, or modules that are being started, marking the point in time where malicious code commences operation. The Evasion (*E.E*) subcategory encompasses techniques that hinder or prevent the analysis of an ongoing attack.
- **Installation** (*I.**): This stage covers Propagation (*I.Pr*), which is all about spreading malware infections and the vertical traversal towards the target. Persistence (*I.Pe*) attacks, on the other hand, attempt to establish a permanent foothold in a system.
- **Command and Control** (*C.**): The C2 channel of an APT is responsible for communication between the victim and the malicious controller. This stage consists of the Download (*C.Do*) category, which includes patching and update mechanisms that alter or expand the original function of malware or exploits, the Directive (*C.Di*) category, which subsumes commands sent via the C2 channel that potentially alter an attack's original purpose, and the Exfiltration (*C.E*) aspect, which includes smuggling out of e.g. previously stolen information.
- **Actions on Objective** (*A.**): These actions encompass the actual victim attack task performed after going through some or all of the above kill chain stages. They again correspond to the CIA triangle of information security, which is also referred to as C, I, and A impact. Every attack action with a suitable CIA impact other than 'none' can be used as $A.*$ action.

Attack Actions

Attack actions represent dedicated hostile behavior, which aims to compromise a target asset or security solution in order to steal information, alter its operational parameters, or negatively influence its availability. The main bulk of these actions and the aforementioned CIA attacks performed by the hostile actor (see Section 6.4.3) is taken from the `Mechanisms of Attack` described in the CAPEC classification. These mechanisms encompass several levels of hierarchy and a description of possible countermeasures – subsequently translated to the defender's arsenal via the NIST Security and Privacy controls (SP 800-53) standard ([146], see 6.4.5 below). In our game model, this mapping links the APT stages with their base detection and success chances to an existing database of usable attacks as well as numerous possible countermeasures. All actions, with their classification into confidentiality, integrity, and availability attacks, are linked directly to the individual mechanisms of attack through the CIA `Impact Rating` provided by the CAPEC standard. Similarly, the mapping between TAON's APT kill chain subcategories and our primary classes of attack is done partly via CAPEC's `Purpose` information: Attack patterns are separated into reconnaissance, penetration, and exploitation categories, which directly map to the kill chain's Reconnaissance, Delivery–Intrusion, and Exploitation stages. The remainder of links (also see Figure 6.8) is assigned manually.

The types of attacks (*primary attacks*) mapped to the [APT kill chain] include (official CAPEC terminology, where differing):

- **Information Gathering *IG*** [Reconnaissance–Identification, Scan] (Analysis): These attacks include interception, finger- and footprinting and various reverse engineering and buffer manipulation tasks aiming at generating a better understanding of a target system.
- **Injection *IN*** [Exploitation–Initialization, Launch]: Injections control or disrupt the behavior of a target or enable the installation and execution of malicious code.
- **Social Engineering *SE*** [Delivery–Deception]: These actions increase the trust in the malicious entity by spoofing legit content or identities through social engineering.
- **State Attack *SA*** [Exploitation–Initialization, Launch] (Time and State): State attacks try to illegally change the state or timing of an application to gain access to otherwise protected resources.
- **Function Abuse *FA*** [Exploitation–Initialization] (API Abuse): The abuse of existing API and protocol functionality typically aims at information exposure, vandalism, degrading or denial of service, or the execution of arbitrary code on the target.
- **Brute Force *BF*** [Delivery–Intrusion, Installation–Propagation, Persistence]: These techniques explore and overcome security measures of the target by e.g. brute-forcing passwords.

- **Illegal Access *IA*** [Delivery–Intrusion, Installation–Propagation] (Subvert Access Control, Spoofing): In this large class of attacks the adversary attempts to bypass access control mechanisms to gain control over a system or data store.
- **Data Manipulation *DM*** [Delivery–Intrusion, Exploitation–All] (Modification of Resources, Protocol Manipulation): Attack actions of this category exploit the characteristics of data structures to gain illegal access or to interfere with the secure operation of a system. They may also alter the system's integrity by manipulating software, files, or otherwise interfere with the operation of an infrastructure.

The two remaining APT stages, Weaponization and Command and Control, are maintained as individual categories independent from CAPEC. Their attack patterns are currently realized through manually defined actions, which will eventually be derived from other information sources.

- **Preparation *PR*** [Weaponization–All]: These attacks describe actions performed on the premises of the attacker to prepare attack tools, research information about the chosen target, and other preparatory tasks invisible to the defender. In the game, weaponization is typically used to generate Insight or reduce the costs of equipment by spending time on e.g. malware coding.
- **Communication *CO*** [Command and Control–All]: C2 traffic is generated whenever a piece of resident malware receives new commands from its malicious operator. PenQuest uses C2 actions to e.g. allow the attacker to change a previously triggered attack action with a reduced risk of detection. See APT kill chain above for more details.

Table 6.6 lists some exemplary attack actions and their in-game properties, as well as their APT kill chain mappings. $\langle ID \rangle$, $\langle PatternClass \rangle$, $\langle Stage \rangle$, and CIA impact ($\langle Mode \rangle$) are retained as part of the model. The remainder is used for linking APT kill chain elements and CAPEC patterns that constitute our attack actions.

Each action comes with a Sophistication requirement specifying the level of knowledge needed to execute an attack. It is directly derived from the `Attacker Knowledge Required` information as identified in CAPEC. The attacker can reduce this prerequisite by employing exploits (see Section 6.4.2).

> *Eventually, Alice's tampering is caught by Bob, triggering the Conflict phase of the game. Alice managed to boost her Insight pool to an impressive 5 points during Stealth phase, which now allows her to be rather aggressive in her approach. Alice still has 9 Initiative points at her disposal (she managed to complete reconnaissance and weaponization during her stealth play), while Bob is left with 11 points for use during the open conflict.*
>
> *Alice wants to take it slowly to maximize her chances. As she can't directly attack her internal prime target, she attempts to take over an exposed asset first. With her initial conflict action, she tries to place her previously weaponized backdoor*

malware on the education organization's web server for easier subsequent penetration. To upload her code, she needs to successfully complete a 'Delivery' action. Alice elects to use a D.I kill chain action, which translates to either a DM, IN, or PH primary attack. Going the data manipulation route (DM), she executes CAPEC attack pattern 105: "HTTP Request Splitting". This particular attack is rated C (medium), I (medium), and A (low). Since Alice is not after information stored on the web server, she opts for an integrity attack. To be successful, she needs to roll a test with her unmodified Sophistication attribute (3) plus Insight modifier (+25%). Thanks to her preparation and her host exploit kit (another +5%), Alice would have a 60% chance to succeed. Unfortunately, Bob's access control policy (−10%) represents a significant obstacle, reducing the final success chance to 50%. Rolling her die, Alice scores a '2' – easy victory. The integrity of the web server is temporarily set to 'highly affected' (2 increments). If unopposed, she would now only need one more 'low' rated attack (which is made significantly easier by the 'Backdoor' malware card she attached to the attack) to completely take over the server, thereby opening the path to her actual victim.

However, Bob still has a chance to react – if he manages to detect the attack. Thanks to his Sophistication (3), his Insight (+10%), his packet filter (+1 SO), and his IDS (+10%), he has a 60% chance to detect Alice's hack. He scores a '5' – and succeeds. The attacker has to unveil her past action to him, giving Bob the opportunity to restore the affected system's integrity and to anticipate Alice's future actions.

Defense Actions

Defender actions directly counter aforementioned attacks and represent the defensive actor's response to hostile activity modeled by PenQuest. In general, each implemented control (i.e. defense action) identified in NIST SP 800-53 translates to one of several defense actions that counter a number of attack actions in the context of our RPG. Some actions directly relate to equipment (especially policies, see 6.4.2) and allow the defender to implement organizational changes that improve security. Others describe the conduct of security scans, the restriction of access to data, spam protection, or the setup of a honeyclient.

In game terms, a successful defense action lowers the rating of the previously executed C, I, or A attack by a certain amount of points. For example, a defense action might lower the effect of a 'high' (3) level availability attack to 'medium' (2), therefore preventing the system shutdown intended by the attacker, while not entirely mitigating the attack.

To bridge the gap between attack actions that are part of CAPEC's vocabulary and the controls of the NIST security standard, we apply a similar, yet more comprehensive and, to a certain degree, automated approach to the one introduced by [91].

First, controls are split into two distinct categories that describe the scope of the security measure:

- **Organization level**: Controls that target the organization level apply to all assets and security solutions currently in play. The NIST standard [146] contains 167 organization-level controls, including policies. To losslessly reduce this amount to a more manageable number, we categorize them into *primary controls* and *defense actions*, both of which can be found below. Organization level controls are designed to cost an increased amount of Initiative (time) to implement.
- **Information system level**: If a control specifically relates to an information system, they can only be applied to one system of the defender's choice once an attack on that system has been spotted (successful detection). There are 57 information system controls in NIST SP 800-53. In the game's context, we support three modes for information system controls derived from the NIST standard:
  1. *Abstracted controls*: With a focus on accessibility, this mode of PenQuest implements an information system version of each of the below primary (organization level) controls. For example, the Account Management (ACM) primary control can simply be used as organization-wide or information system variant.
  2. *Related controls*: For increased modeling accuracy, we can utilize NIST's `Related Controls` associated to below primary categories as system-level equivalent. See the Account management (ACM) primary control for an exemplary list.
  3. *Control enhancements*: Players can also opt to use NIST `Control Enhancements` for each of the primary controls as information system-level defense measure, provided the primary control is not already an information system control (marked by an asterisk). This mode can be adapted to have control enhancements serve as sole defender action set, omitting organization level controls (except policies) entirely. PenQuest was implemented and tested using this mode.

In the next step, we identify key controls that can be used to accurately depict a class of actual countermeasures. While NIST provides 17 families with a total of 224 controls, these are typically too high level in their categorization to serve as links to the more technical attack patterns used as attack actions. If we would use families directly, the game's rules would have to be adapted to connect each individual control to an attack action, which does not scale well and negatively impacts the applicability of the model to other domains. We therefore specify a number of *primary controls*, which are the polar opposite of the primary attacks defined in Section 6.4.5:

- **Information Leakage Protection *IL*** (counters IG): This primary control prevents information leakage through diligent configuration and data protection. It is associated with NIST's Configuration Settings (COS), Boundary Protection (BOP), and Cryptographic Protection (CRP) defense actions (see below).

- **Context Protection *CP*** (counters *IN*): As control against injection attacks, this category protects from undesired functionality that lets the attacker break out of the current system, communications channel, or application. Associations: Security Engineering Principles (*SEP*), Malicious Code Protection (*MCP*), and Information System Monitoring (*ISM*).
- **Awareness *AW*** (counters *SE*): This group of controls helps the defending organization to raise awareness for social engineering attacks of any kind, including spear phishing and physical intrusion attempts. Association: Role-based Security Training *RST*.
- **State Protection *SP*** (counters *SA*): Protecting the state of an information system is a vital task spanning several groups of actions, ranging from backup systems to integrity protection measures and status monitoring. It is associated with Configuration Change Control *CCC*, Configuration Settings *COS*, Contingency Plan *COP*, Incident Handling *INH*, Nonlocal Maintenance *NOM*, and Information System Monitoring *ISM* actions.
- **Function Integrity *FI*** (counters *FA*): Similarly, function integrity controls make sure that the available functionality (API, commands) of an application are not in any way abused. NIST associations include: Configuration Change Control *CCC*, Security Engineering Principles *SEP*, Malicious Code Protection *MCP*, and Information System Monitoring *ISM*.
- **Authentication Protection *AP*** (counters *BF*): This control group is primarily concerned with managing authenticators such as passwords and tokens. Associated controls: Remote Access *REA*, Authenticator Management *AUM*.
- **Access Control *AC*** (counters *IA*): As a main countermeasure to a wide range of intrusion attacks, the access control family subsumes account management, enforcement strategies, and various access-related policies. Associated controls are: Account Management *ACM*, Access Enforcement *ACE*, Continuous Monitoring *COM*, Least Privilege *LEP*, and Remote Access *REA*.
- **Data Integrity *DI*** (counters *DM*): Maintaining the integrity of data is one of main tasks of information security. In our gamified model, this primary controls includes: Contingency Plan *COP*, Incident Handling *INH*, Cryptographic Protection *CRP*, Malicious Code Protection *MCP*, and Information System Monitoring *ISM*.
- **Security Intelligence *SI*** (counters *PR*): Preparation for an attacks works both ways: Potential victims use intelligence techniques to stay up-to-date with threats and prepare their systems for any eventuality.
- **Communications Security *CS*** (counters *CO*): The flow of information between internal and external system is a likely target for attack. In this group, we combine the following controls: Information Flow Enforcement *IFE*, Boundary Protection *BOP*, Continuous Monitoring *COM*, Cryptographic Protection *CRP*, and Information System Monitoring *ISM*.

ID*	Name*	⟨*Cat.*⟩*	⟨*ControlClass*⟩	⟨*ActionClass*⟩	Related controls*	Control enh.*
AC-2	Account management	Org.	AC	ACM	AC-10, AU-9, IA-2, IA-8	AC-2 (1)..(13)
AC-3	Access enforcement	Sys.	AC	ACE	AU-9	AC-3 (1)..(10)
AC-3 (3)	Mandatory access control	Sys.	AC	ACE	AC-25, SC-11	n/a
SI-3	Malicious code protection	Org.	CP,SP, FI,DI	MCP	SC-26	SI-3 (1)..(10)

Table 6.7: Excerpt from the current pool of PenQuest defense actions. Columns marked with asterisk (*) identify information directly mined from NIST controls. Related controls only list information system level controls that are not in the primary category.

We now map these controls to the following representative *defense actions*, which were automatically extracted (see Section 6.5 for details) from NIST's control catalog [146] by assessing their strength of association described through their official `Related Controls` property. The official NIST control ID and information system controls are separately identified. Multiple mappings specify that several countermeasure classes and its related controls are effective in the respective scenario. The defense action listed below exemplarily includes related information system controls and control enhancements, which are finer-grained countermeasures within its context. Please refer to the Appendix of Luh et al. [195] for a full lost of control-to-action mappings.

- **Account Management *ACM*** (AC-2 → AC): Establishes conditions for group and role membership, specifies authorized users and access authorizations (i.e., privileges), creates, enables, modifies, disables, and removes information system accounts, and monitors their use.
  - Related (non-primary) information system controls: Concurrent session control (AC-10), Protection of audit information (AU-9), Identification and authentication for organizational (IA-2) and non-organizational users (IA-8).
  - Control enhancements: Automated system account management, Removal of temporary/emergency accounts, Disable inactive accounts, Automated audit actions, Inactivity logout, Dynamic privilege management, Role-based schemes, Dynamic account creation, Restrictions on use of shared/group accounts, Shared/group account credential termination, Usage conditions, Account monitoring/atypical use, Disable accounts for high-risk individuals.

The key component to finalizing the game model is the comprehensive mapping of simplified NIST families and controls (defense actions) to specific CAPEC mechanisms of attack (attack actions). See Section 6.5 for details about this process. In Table 6.7, we list some exemplary defense actions and their offensive counterparts.

> *With Alice's attack cycle complete, it is now time for Bob to deploy his countermeasures. As he earlier spotted an integrity attack targeting his web server,*

he assumes that Alice is trying to deface his institution's Internet presence. In response, Bob can now either implement a system-level 'data integrity' (DI) control for 1 Initiative point to counter the data manipulation (DM) attack, or an organization-wide control for 2 points. The latter would not complete within the same round, which poses too great a risk for Bob. He decides to do what he can and restore system integrity from 'highly affected' (2) to 'affected' (1). Depending on the game mode, he could either use an abstracted version of the COP, INH, CRP, MCP, or ISM controls, or implement related controls or control enhancements.

In this example, we use abstracted control enhancements: Bob checks each of the defense actions at his disposal and decides to use an enhancement of MCP (SI-3), namely SI-3(7): "Malicious code protection: Nonsignature-based detection". The default success chance of such actions is derived from twice his base Sophistication (6), Bob's Insight (+10%), and all equipment boosting either of the two. In our case, Bob is granted another point of Sophistication because of his packet filter, for a total defense success chance of 80%. Bob rolls a D10 and succeeds: The level of compromise of his web server is decreased by one increment to 'affected' (low) – a distinct setback for Alice.

The game will now continue until the attacker achieves her goal or runs out of Initiative, ultimately deciding the winner.

6.5 Data Mapping

In this section, we specify the mapping of external data to the various classes and categories that are part of our model. This includes the formal link between actions and events, the mapping of the APT kill chain to CAPEC attack patterns, the CAPEC to CVSS mapping, and the association of controls to defense actions. Using below information, it is possible to easily extend or adapt the game system to new or updated scenarios in cyber-security and beyond.

6.5.1 Actions to Events

Each action available in the PenQuest rule system can be modeled using the structure introduced in Section 6.3. The possible link between in-game actions and real-world events is an integral part of the model. We below exemplify this mapping using the star graph anomaly detection system that is at the heart of the AIDIS endpoint protection system. The system is built for detecting and interpreting abnormal Windows OS behavior expressed through sequences of kernel events defined as $G = (U, V, E)$, where U and V are nodes (parametrized event type) and E is the respective edge (type of operation). Specific attacks can be learned by monitoring process activity and comparing it to a pre-established baseline of known process behavior. In a simplified fashion, a number of events contributing to an anomaly could look like the ones listed in Table 6.8.

Start node (U)	End node (V)	Edge (E)
process-shell.exe	process-drop.exe	start (3)
process-drop.exe	image-library.dll	load (1.5)
process-drop.exe	registry-HKLM/Software/.../WindowsFirewall	open (0.25)
process-drop.exe	registry-DWORD(EnableFirewall=0)	add (0.75)

Table 6.8: Example event sequence describing the process of disabling the Windows Firewall (CAPEC-207: "Removing important client functionality").

To append the data to the model, we use the PenQuest game model (Chapter 6.3.2) to transform the list of events to a simple instance of the $\langle Event \rangle$ class of an action X, shortened here to two sequential events:

$$
\begin{aligned}
&Event = \langle \\
&\quad \langle Type = Anomaly \rangle \\
&\quad \langle Time \langle Start = 16.53.661, End = 16.53.729 \rangle \rangle, \\
&\quad \langle Score \langle Deviation = 112.4, Threshold = 16.0 \rangle \rangle, \\
&\quad \langle Sequence = 1 \rangle, \\
&\quad \langle Parent = \text{“}shell.exe\text{”} \rangle, \\
&\quad \langle Operation = process\_start \rangle, \\
&\quad \langle Argument = \text{“}drop.exe\text{”} \rangle \\
&\quad \langle Sequence = 2 \rangle, \\
&\quad \langle Parent = \text{“}drop.exe\text{”} \rangle, \\
&\quad \langle Operation = image\_load \rangle, \\
&\quad \langle Argument = \text{“}library.dll\text{”} \rangle \rangle
\end{aligned}
$$

Since AIDIS classifies anomalies by their CAPEC attack pattern, the respective information can easily be added to the action definition:

$$
\begin{aligned}
&AttackPattern = \langle \\
&\quad \langle Purpose \langle Exploitation = T, Obfuscation = F, \\
&\quad Penetration = T, Recon = F \rangle \rangle, \\
&\quad \langle Impact \langle C = high, I = high, A = low \rangle \rangle, \\
&\quad \langle ID = 207 \rangle \rangle
\end{aligned}
$$

This `Purpose` information of CAPEC is then used to establish the link to our abstracted *primary attacks*, which is discussed below.

This simple interface makes it easy for analysts to add their own data to the game model. While our game rules use the CAPEC example, PenQuest remains flexible: By replacing the *AttackPattern* class definition, the level of abstraction and vocabulary can be freely specified – ranging from the discussed OS events to high-level behavior such as 'Set Fire to House'. See Section 6.6 for a full example in the context of a real IDS anomaly.

6.5.2 Kill Chain to Attack Patterns

We use Hutchins et al.'s cyber kill chain [133] as foundation for the high-level view on our model. For further granularity, we expanded the 7 stages to a total of 19 subcategories that largely adhere to our APT ontology design introduced with TAON [188]. To link kill chain elements to CAPEC attack patterns, we introduced the concept of *primary attacks*, which were abstracted from CAPEC's `Purpose` classes and assigned attacked patterns to kill chain categories. Figure 6.8 depicts the mapping.

Since the list of attack patterns in CAPEC is partially incomplete or offers too little information in terms of abstraction required for a model, we only considered patterns of 'standard' and 'meta' abstraction level – the 'detailed' class was omitted in this initial iteration of the game. In addition, we opted to remove deprecated attacks and focus only on stable and draft patterns found in version 2.11 of CAPEC[1]. Incomplete patterns with no purpose classes were disregarded as well. Overall, this selection retained a total of 65 representative patterns out of 516. The remainder can easily be added at a later point.

6.5.3 Attack Patterns to Vulnerabilities

Vulnerabilities are a key component of any intrusion scenario. In order to automatically map CAPEC attack patterns to CVSS vulnerabilities, we took a closer look at these and other related MITRE information exchange standards. Ultimately, we use the `Related Weaknesses` information provided by CAPEC to map each pattern to specific weaknesses represented by the Common Weakness Enumeration (CWE) list. CWE *"provides a common language for describing security weaknesses in architecture, design, or code"*[2]. For example, CAPEC ID 1 ("Accessing Functionality Not Properly Constrained by ACLs") is related to CWE ID 276, 285, 434, etc.).

To close the gap between weaknesses and the more concrete vulnerabilities, we use open source information to map CWE entries to their Common Vulnerabilities and Exposures (CVE) counterpart. For example, CWE-276 ("Incorrect Default Permissions") contains the vulnerabilities CVE-2005-1941, CVE-2002-1713, CVE-2001-1550, CVE-2002-1711, CVE-2002-1844, CVE-2001-0497, and CVE-1999-0426. Armed with this ID, the specific vulnerability can be looked up in the National Vulnerability Database (NVD[3]), where a specific CVSS score is assigned. This multipartite score is the basis for the requirements and impact calculations used in our model (see $\langle Enabler \rangle$ in Section 6.3). In our example, the score for CVE-2005-1941 can be easily retrieved[4].

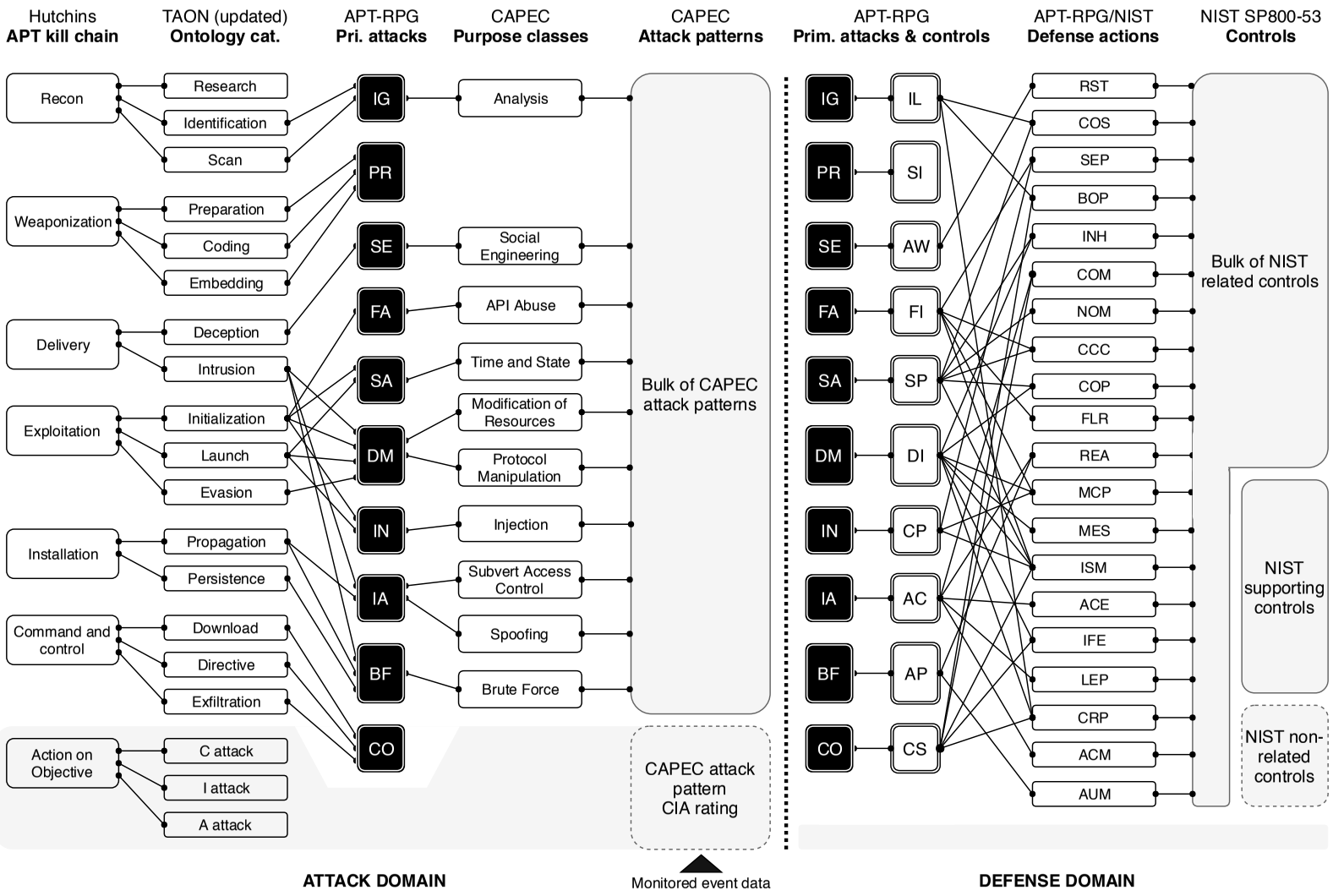


Figure 6.8: Mapping of NIST controls to CAPEC attack patterns via extended APT kill chain. The introduced link categories based on CAPEC are highlighted in black.

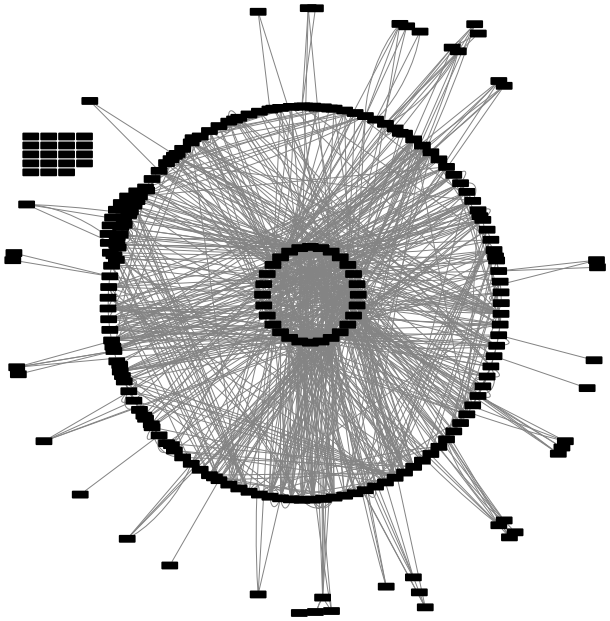


Figure 6.9: Relations between NIST controls rendered in Cytoscape [286]. The inner circle represents controls with a degree of $\geqslant 20$. Elements beyond the outer circle have a degree of 1, while the isolated nodes to the left are not linked to any other controls (orphan controls).

6.5.4 Primary Controls to Defense Actions

The mapping of controls to defense actions and primary controls was automatically computed from the NIST SP 800-53r4 standard [146]. To achieve this, we extracted all links of relationship between the individual controls and modeled them as a graph. We subsequently separated each control into one of three categories by their level of degree. Controls with a degree $d >= 20$ were defined as parent *defense actions* (see Figure 6.8). Controls with degree $d < 20$ and $d >= 1$ form the bulk of related controls which operate on information system and/or organization level and translate to the remainder of the defense strategy set for the respective parent action. The model can be further extended by adding control enhancements that, in turn, are associated with numerous parent and related controls. Figure 6.9 depicts the relationship between all 224 NIST controls. General measures (NIST suffix *-1) were converted into policies, which are part of the $\langle Disablers \rangle$ class (see Section 6.4.2 for a complete list).

For the mapping between defense actions and primary controls, a natural language approach has been investigated. By combining four IT and information security glossaries (Gartner[1], Techopedia[2], NIST-IR 7298 [157], and the North Carolina statewide glossary of information technology terms[3]) into a list of reverse stopwords (i.e. the remainder of words were removed from the corpus), we attempted to automatically link

[1]https://capec.mitre.org
[2]https://cwe.mitre.org/about/index.html
[3]https://nvd.nist.gov
[4]https://nvd.nist.gov/vuln/detail/CVE-2005-1941
[1]https://www.gartner.com/it-glossary/
[2]https://www.techopedia.com/dictionary
[3]https://it.nc.gov/document/statewide-glossary-information-technology-terms

CAPEC pattern and NIST control descriptions using Quanteda for R [30]. However, the difference in terminology and writing style rendered the approach too inaccurate for practical implementation in this particular case. Instead, we opted to create primary controls (see Section 6.4.5) and directly assign them to the aforediscussed defense actions. Figure 6.8 depicts the complete mapping.

6.6 Preliminary Evaluation

In this section, we briefly introduce our first physical prototype of the PenQuest RPG and present the quantitative and qualitative findings of our initial test games designed to determine the suitability of the gamified model for (awareness) education and threat explication. We also take a closer look at data-to-model mapping as well as strategy set distribution between the attacker and defender to measure the model's completeness. The discussion about future scenario simulation and IT-enabled automation can be found in Section 6.7.

6.6.1 Experimental Setup

Prototype

Our first operational prototype uses a physical approach to present the game and its components to a predominantly information security audience. Actions and equipment are designed as cards containing all necessary information ranging from categories, requirements, and game impact. Progress tracking for asset compromise, APT kill chain traversal, and success modifiers (e.g. Insight, success chance) is realized through physical tokens placed on a printed game board. Similarly, levels of compromise, actor attributes, credits, and attack objectives are tracked using a combination of cards and tokens. Figure 6.10 depicts an early version of the game board. Next to visualizing attack vectors and asset dependencies, it provides players with a kill chain tracker for planning their attack and defense.

A total of 100 attack actions and 70 defense actions were prepared for the test games. Existing CAPEC patterns and NIST control enhancements were complemented by a small number of placeholder actions for the weaponization and C2 phases of the APT kill chain, which are not currently covered by CAPEC. In addition, we designed 50 unique attack tools (enablers) and a total of 58 disablers, 18 of which represent organization-wide policies. This was complemented by a set of 25 representative vulnerability exploits and a total of 18 fixes substituting the CVE component of the game. In future automated simulation games that do not need to meet the same level of accessibility requirements, vulnerabilities will be retrieved directly from the NVD.

IT support for the current implementation already exists in the form of the WF-net validator introduced in Section 6.3.3 as well as a PoC prototype for a two-player browser app that incorporates actor creation and basic game board interaction. This does not

mean that PenQuest's physical iteration is limited in terms of overall functionality, however: the game simply requires a human moderator to enforce some of its more advanced rules and keep track of point tallies – akin to many commercial strategy games and moderated red-team exercises [260].

Questionnaire

In order to evaluate the physical game prototype and the model's suitability for representing APTs we invited a number of test persons from a corporate and educational IT security background and presented them with both an introductory and a concluding questionnaire. Next to age, gender, and specific level of security expertise we asked participants to answer a number of simple questions about IT security topics to score their current level of knowledge. Answers to the following questions were ranked from 'strongly disagree' (0 points), 'rather disagree' (1 point), 'rather agree' (2 points), to 'strongly agree' (3 points):

1. I know how cyber-attacks typically play out
2. I am familiar with the APT kill chain
3. I am familiar with the NVD and its scoring system (CVSS)
4. I am familiar with the concept of IT system vulnerabilities
5. I am familiar with CAPEC or similar attack pattern schemas
6. I am familiar with standards like ISO 2700x and NIST 800-53
7. I am familiar with information security best practices
8. I am familiar with IT network topology
9. I am familiar with common types of malware
10. I am familiar with the functionality of common hacking tools
11. I know how an organization can defend against cyber-attacks

These questions were repeated after one play-through in order to evaluate lessons learned. In addition, the subjects graded the game's suitability for education and threat modeling in e.g. risk assessment scenarios. Specifically, we asked questions about accessibility (learning curve, handling of the prototype, abstraction level of attack and defense actions), realism (applicability to real-world scenarios, scope of actions/equipment), game balance (attacker/defender equilibrium in same-Sophistication games), and awareness benefit (personal takeaway, educational effect). For threat modeling, subjects were asked to rank the model's suitability for awareness building and threat representation (abstraction level, real-world application, incident representation). All games were documented to protocol action dynamics and session outcome. Note that testers playing more than one game were assessed only once and that none of them were involved in the creation of PenQuest.

Expert Interviews

All participants of at least 'professional' IT security level (see demographics below) were also given the opportunity to provide individual textual and face-to-face feedback. While general comments were encouraged, we asked the remaining 7 practitioners to focus on criticism in the areas of game accessibility, balance, and design. To assess the underlying model, we also provided them with a copy of PenQuest's full documentation (base, game, and rule model) as well as with the game's ruleset in scientific and instruction guide format and requested specific feedback on each of the aspects. The results of these interviews are part of the qualitative evaluation below.

IDS Data

To demonstrate IDS data-to-model mapping, we used classified anomaly events of our own AIDIS system as input. The system provides automated classification of anomalies as belonging to a specific CAPEC pattern, through which it becomes possible to directly link extracted behavioral data to the PenQuest meta model for further semantic enrichment, interpretation, and mitigation planning.

For evaluative demonstration, we use the CAPEC class with the most events while boasting a low misclassification rate, as discussed in Chapter 7. We specifically regarded CAPEC-112 [1], 'Brute Force', with a total number of 380 process anomaly reports and a misclassification rate of 0% in the context of AIDIS's operation. The concrete mapping is discussed as part of the qualitative evaluation below.

6.6.2 Quantitative Results

In this subsection, we quantitatively evaluate the test games conducted with a number voluntary participants. While the number of players is arguably limited, the results still help to get an impression for the practical applicability of PenQuest's game component in awareness building scenarios. In addition, we enumerate the rule system itself to assess its coverage of various attack categories.

Test Games

Of the 8 full test games played (in addition to approx. 12 partial sessions), 5 ended in victory for the attacker. The average session lasted for 7 Initiative rounds. Initial game setup and the first-time explanation of the rules to the participants new to PenQuest took an average of 45 minutes, while the actual playing session concluded in around 115 minutes. Stealth phase lasted an average of 1.88 rounds. On the kill chain, attacking players rarely exceeded the Exploitation ($E.*$) stage that marks the conclusion of a 'quick mode' game. In terms of scenario, the play sessions i.a. encompassed 'Operative

[1] https://capec.mitre.org/data/definitions/112.html

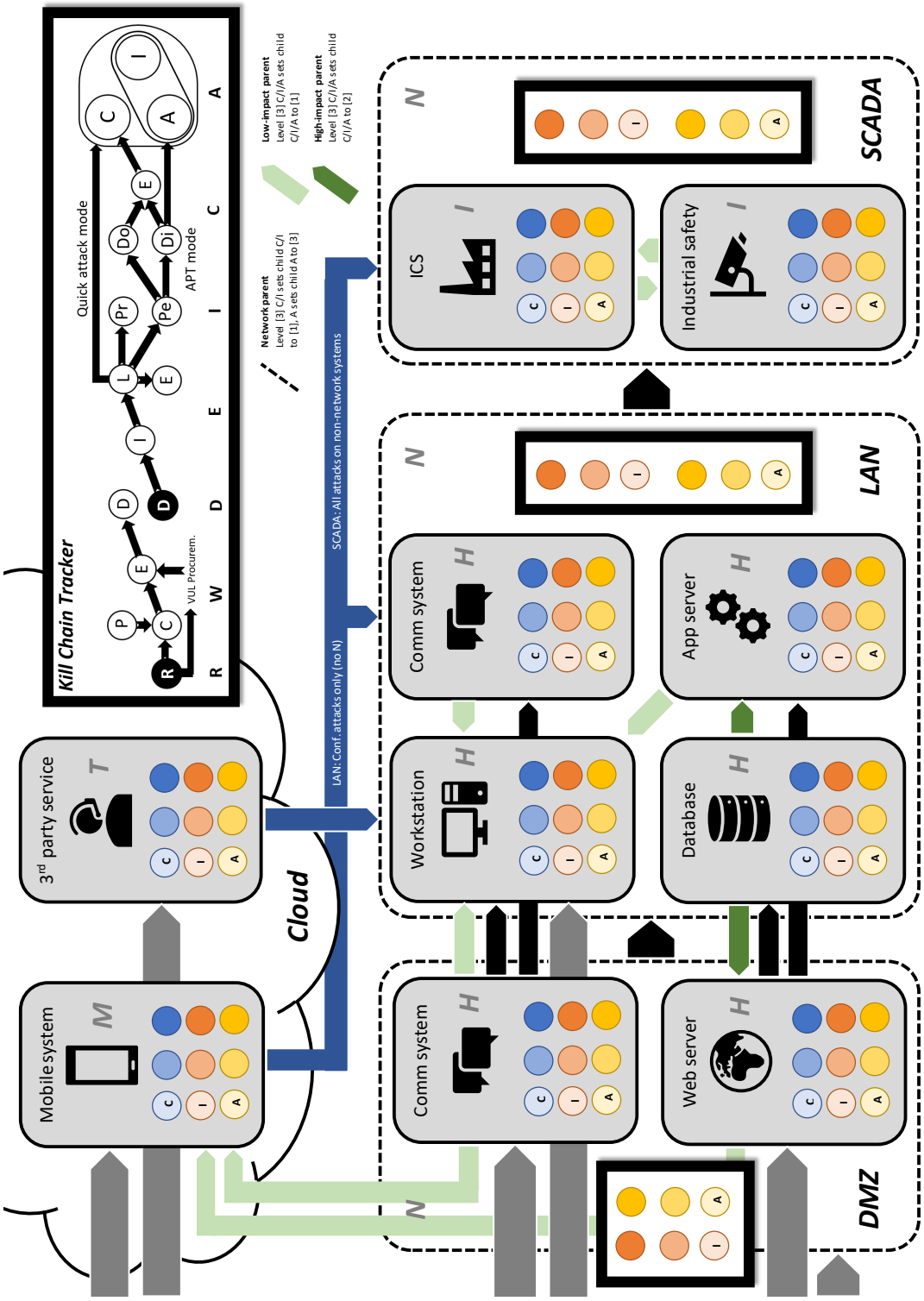


Figure 6.10: Prototype game board used by players. Current (known) levels of compromise, the progress along the kill chain, and the attacker's primary goal are marked here. Current Insight and Initiative is tracked using tokens. The remainder of the game utilizes a total of 359 cards.

	Int1			Int2			Pro1			Pro2			Pro3			Exp1			Exp2			Exp3			Exp4			
Qn	B	A	+	B	A	+	B	A	+	B	A	+	B	A	+	B	A	+	B	A	+	B	A	+	B	A	+	**Mean**
Q1	1	2	1	1	2	1	2	2	0	2	2	0	2	2	0	3	3	0	2	3	1	2	2	0	2	2	0	0.33
Q2	0	2	2	0	1	1	2	3	1	1	2	1	1	2	1	2	2	0	2	3	1	3	3	0	1	2	1	0.89
Q3	0	0	0	0	1	1	2	2	0	0	1	1	1	2	1	1	1	0	1	0	-1	2	2	0	3	3	0	0.22
Q4	0	1	1	0	1	1	2	2	0	3	3	0	2	2	0	3	3	0	3	3	0	2	3	1	3	3	0	0.33
Q4	0	2	2	0	1	1	1	2	1	1	1	0	1	2	1	0	0	0	0	0	0	2	3	1	2	2	0	0.67
Q5	0	1	1	0	1	1	2	2	0	2	2	0	3	3	0	1	1	0	0	0	0	3	3	0	1	2	1	0.33
Q6	1	2	1	1	2	1	2	2	0	3	3	0	3	3	0	3	3	0	2	3	1	3	3	0	2	2	0	0.33
Q7	0	2	2	0	1	1	2	2	0	3	3	0	2	2	0	3	3	0	2	3	1	2	2	0	3	3	0	0.44
Q8	2	3	1	1	2	1	2	2	0	2	2	0	2	2	0	3	3	0	2	3	1	3	3	0	3	3	0	0.33
Q9	0	1	1	0	1	1	2	2	0	1	1	0	1	2	1	3	3	0	2	3	1	2	2	0	3	3	0	0.44
Q10	1	1	0	0	1	1	2	2	0	2	2	0	2	2	0	3	3	0	2	2	0	3	3	0	3	3	0	0.11
Sum			12			11			2			2			4			0			5			2			2	4.44

Table 6.9: Total knowledge gain per participating player. The score is derived from a self-assessment conducted once before playing PenQuest (B) and a second time thereafter (A). There are 4 answer categories per question, ranging from 'strongly disagree' (0) to 'strongly agree' (+3). Each point of improvement ('+' column) represents the knowledge gain of the individual player, ranging from 0 to 3. The sum in the lowermost row designates the overall knowledge gain per participant.

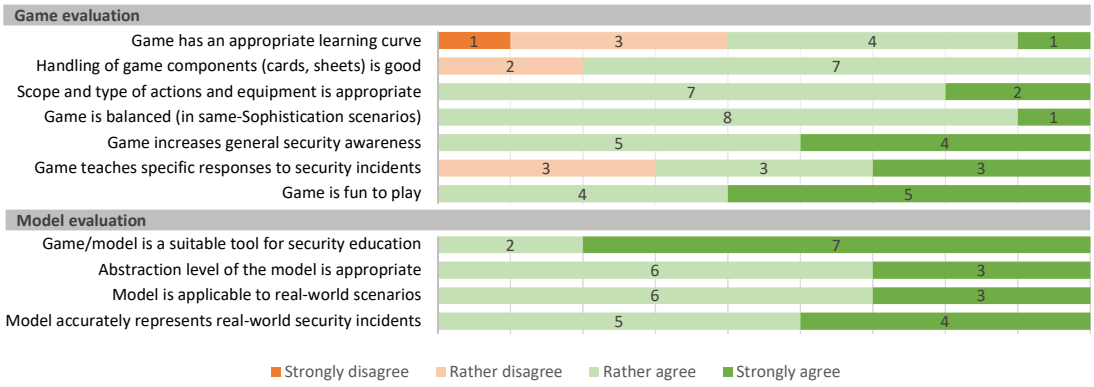


Figure 6.11: Player feedback by question, ranging from 'strongly disagree' to 'strongly agree'. While most players would use the game as part of an awareness program, almost half of them find the learning curve to be rather steep.

vs. Military', 'Protester vs. Education', 'Infiltrator vs. Infrastructure', 'Thief vs. Manufacturing', and 'Raider vs. Services Sector' scenarios.

Demographically, most players (7) of the total 9 were male, aged 26 to 35 years. Dominant occupations were university lecturer/researcher, company employee (5 and 3 participants respectively), and 1 student, while the level of knowledge was evenly spread between 'Operator/Specialist' and 'Executive/Management' (4 and 4). One 'Assistant/Intern' participated in the testing. Occupation-wise, the I(C)T/informatics sector was represented the most (7 participants), followed by marketing/media (1) and general education (1). All of the players claimed to have at least intermediate experience in the area of IT security. Of the 9 participants, 4 ranked themselves as experts (identified as *Exp∗* in Table 6.9), 3 as professionals (*Pro∗*), and 2 as intermediate (*Int∗*). Most players (5) did not have prior experience with learning games. Only one person designated herself as moderately experienced in serious gaming.

According to the returned questionnaires, the greatest learning effect was achieved in the areas of the APT kill chain, CAPEC attack patterns, network topology, and the functionality of hacking tools. Interestingly, the area of CAPEC was also the one where many participants (4 out of 9) showed no knowledge gain after playing the game. This is likely owed to the fact that most games were not contentually moderated – most of the time, the game master focused on explaining mechanisms and rules instead of information security aspects. Either way, understanding attack patterns has been identified as an area in high need of impartation beyond the single line of explanatory text currently printed on the card.

Non-experts benefited the most from playing the game, reporting the highest gain in domain knowledge. Experts still claimed a minor increase in topical insight in 3 out of 4 cases. See Table 6.9 for an overview.

When asked to grade their personal experience, all the players stated that it was overall positive. All participants agreed that the game showed significant promise for educational use in a higher education or company environment. The question as to whether the "Game/model is a suitable tool for security education" was answered positively most often, with an average score of 2.78 out of 3, followed by the statements "Game is fun to play" (2.56), "Game increases general security awareness" (2.44), and "Model accurately represents real-world security incidents" (2.44). In terms of learning curve, answers were the most controversial (1.56 out of 3), indicating a need to further improve accessibility for players unfamiliar with serious games. Figure 6.11 depicts the results of the final questionnaire.

Strategy Set Distribution

The distribution of actions between the attack and defense domains is both an indicator for control effectiveness (where few controls counter a large number of attacks) and model completeness. In Figure 6.12, we show the distribution of attack/defense actions per primary attack/control. In the current prototype, state attacks (SA) were identified as underrepresented due to a lack of CAPEC patterns meeting the requirements specified in Section 6.5.2. This can be partially remedied by adding 'detailed'-level controls to the model.

In terms of APT kill chain coverage of the game, the CIA triangle is represented with a nearly identical number (60 to 66 each) of available attacks corresponding to the three information security factors. With the exception of availability attacks where 'low'-rated attacks are predominant, most of CAPEC's utilized patterns describe 'high' impact attacks, followed by 'medium' attacks. Stage-wise, most actions (120) belong to the Exploitation phase, with around 45 patterns linked to Delivery and Reconnaissance. The remainder of the kill chain is represented by everything between 16 and 39 CAPEC and placeholder pattern attacks. See Figure 6.13 for a full breakdown.

Figure 6.14 quantifies the number of attack patterns per defense action category. Unlike the strategy set distribution, these numbers encompass all currently available

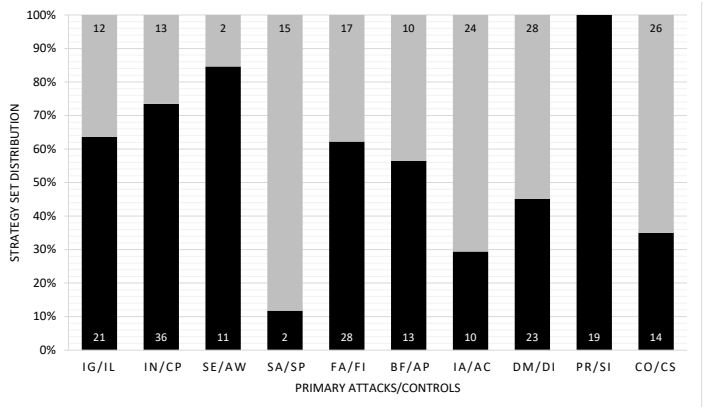


Figure 6.12: Distribution of attack/defense actions per primary attack (black) and primary control (gray). Some attack actions (such as on-premises weaponization (PR)) do not have counterparts.

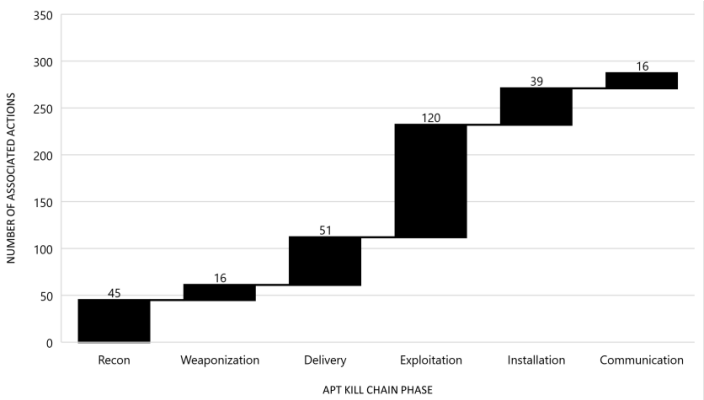


Figure 6.13: APT kill chain coverage through available attack actions. The 'Action on Objective' stage is integrated into the other categories by means of the game model.

categories as per the data mapping specified in Sections 6.5.2 and 6.5.4. Here, the number of attack patterns countered by each defense action is largely well distributed. Media Storage (MES), Security Engineering Principles (SEP), and Cryptographic Protection (CRP) controls are not enhanced by more granular NIST countermeasures, making them particularly effective if employed at organization level. At the same time, there are only few attack patterns that can be countered by Nonlocal Maintenance (NOM), which is less concerned with deliberate attacks and more with system upkeep.

The distribution of the categories clearly shows where current standards offer fewer, if broader – and potentially more effective – guidelines in terms of mitigation. For many technical attacks NIST suggests only a few countermeasures without going into detail with extended controls. Next to above examples, this includes anti-malware solutions, configuration management, and (software) flaw remediation. Similarly, attack patterns are not equally easy to come by and tend to describe some types of attacks over others. Especially system state manipulation, spoofing, and the subversion of access control mechanisms are harder to specify in significant quantity than e.g. injection attacks.

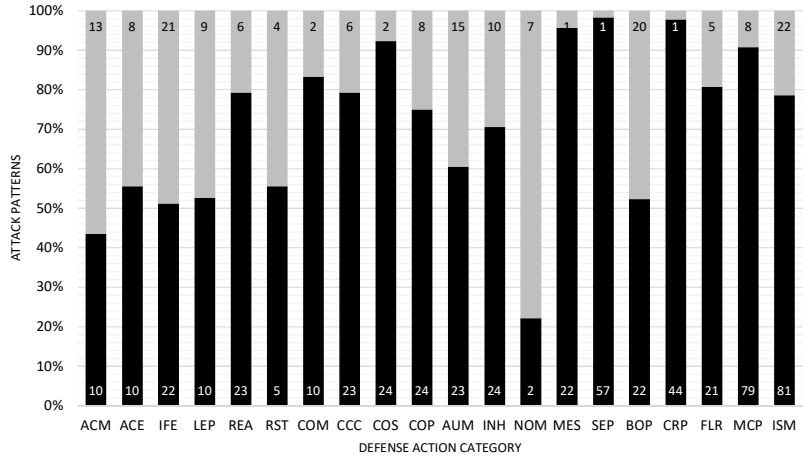


Figure 6.14: Attack patterns (black) per defense action category (gray). This chart encompasses all actions and categories currently available for game development.

6.6.3 Qualitative Results

When asked to provide additional textual and oral feedback, security practitioners contributed valuable insight and suggested a number of improvements to both model and game. A summary can be found below. Lastly, we sketch how the mapping of IDS data to the model works in practice.

Game

In stage 1, we collected feedback aimed directly at the game and its prototype implementation. This information is particularly important for the ongoing development of PenQuest's digital iteration and primarily addresses three topics of interest: accessibility, balance, and design.

Accessibility – One of the major points raised in the expert discussion was that the current prototype aims at a target audience with at least a Bachelor-level degree in computer science or IT security. While terms and explanations used in the game correspond to the current business practice, the entry level for students or employees of other areas is too high. Several participants suggested that future versions of PenQuest – provided they also seek to target non-IT players – should paraphrase security concepts in simpler language and provide the original information as footnote pointing to an external reference.

It was also noted by a tester that people unfamiliar with complex strategy games might be overwhelmed by the amount of rules. In contrast, participants who plays board games regularly described PenQuest's rule set as "about average in difficulty" and added that it is common for such games to require 2 or more playthroughs to get a full grasp of the more advanced mechanics. Most of the interviewed stated that having a digital system for calculating system compromise and success probabilities would be

beneficial in terms of accessibility.

Balance – Tester feedback was mostly in line with the results of our quantitative evaluation of PenQuest's strategy set distribution. While some actions were perceived as more effective than others, there were no trump cards that enabled an easy victory. Without exception, the participants agreed that the slight bias towards the attacking actor is both realistic and entertaining – and helps transport the message of awareness.

The majority of experts showed themselves surprised that even though the model uses real-world data sources for its loss and gain scores, the resulting game is naturally balanced. It was added, however, that this equilibrium would likely be lost without the limiting factors of Initiative and Wealth, which temporally and financially constrain the modeled adversarial campaign.

Design – Since the visual design of the game is not the focus of this article, we only briefly summarize tester feedback: While most game elements were deemed as practical and comprehensible, players suggested various improvements to iconography and game board design. In particular the CIA impact of attack actions and the representation of the APT kill chain were criticized as not immediately self-explanatory.

Model

The second stage of the qualitative feedback process revolved around the model on which the game is built. After playing at least one test game and having been given a detailed introduction to the base, game, and rule models as well as the game's resulting ruleset, participants were asked to provide criticism from their perspective as experienced security practitioners.

Base model – Testers agreed that PenQuest's base model serves well to introduce the core concepts of the gamified system and defines all terms needed to understand layer and component interplay. It was suggested, however, that future iterations of the model could benefit from additional detail, especially for the 'knowledge' and 'configuration' aspects found within the model's information layer. One expert recommended that the concept of vulnerabilities could be incorporated directly into the base model, yet admitted that detaching them from the 'equipment' definition would add an undesired layer of complexity.

Game model – Generally, PenQuest's use of attack and defense classes to abstract attack patterns and controls was lauded by the testers. Most experts emphasized that the approach helps to better understand the model's mapping procedures and enables security practitioners to plan an appropriate defense against threats even without understanding all the specifics of each individual attack or control.

Scrutinizing feedback included the aforementioned vulnerabilities and fixes, which are currently part of the $\langle Enabler \rangle$ and $\langle Disabler \rangle$ classes of action X. It was suggested by an expert to consider creating a class of its own for both, something that will be investigated for future versions of the model.

When discussing the game model's $\langle Event \rangle$ class, several of the participants independently confirmed that it provides the means to interface PenQuest to not only IDS data, but to established threat intelligence languages as well. This link and the construction of an ontology based on our meta model will be thoroughly explored in future research.

Rule model – All participants agreed that the game's core principles aptly capture the essence of the problem. The zero-sum component for tallying system compromise was deemed a suitable abstraction for the process of attacking an asset with a certain goal in mind.

Game rules – The discussion of the rules resulting from the above model components took up most of the allotted interview time. Testers predominantly agreed that the rules are a good compromise between a realistic simulation and an awareness-building game that is also entertaining its players. Feedback mostly related to future work and additions to the game, which included awarding situational modifiers depending on which $\langle AttackActor \langle Motivation \rangle \rangle$ the attacking player has chosen. It was stated that PenQuest would also benefit from replacing generic equipment with real-world appliances and tools, while another expert noted that such a move would be akin to advertising and should be considered carefully.

Several testers were enthused by the fact that the current, abstracted topology could be easily expanded to mirror more complex networks. At the same time, they suggested that threat simulation based on PenQuest would benefit from a formalized 'topology creation mechanism' that helps prevent design errors in terms of attack vector and asset dependency. Two participants suggested the use of a more streamlined version of the APT kill chain for modeling the typical sequence of attack actions. While its current level of detail was appreciated and deemed useful for information security audiences, the testers argued that a simpler representation without sub-stages would be beneficial for the average player.

Data Mapping

Concluding the qualitative evaluation, we take a look a specific use case where data captured by an IDS is mapped to the PenQuest model for semantic enrichment and mitigation planning. For this purpose, we first use PenQuest's $\langle Event \langle Type = Anomaly \rangle \rangle$ notation of X. The exemplary event sequence extracted by AIDIS has a time range of $\langle Time \langle Start = 0, End = 10 \rangle \rangle$. As seen below, each $\langle Operation \rangle + \langle Argument \rangle$ pair with $\langle Parent = svchost.exe \rangle$ is appended in sequence.

$$
\begin{aligned}
&Event = \langle \\
&\quad \langle Type = Anomaly \rangle \\
&\quad \langle Time \langle Start = 0, End = 10 \rangle \rangle, \\
&\quad \langle Score \langle Deviation = 62.7, Threshold = 45.8 \rangle \rangle, \\
&\quad \langle Sequence = 1 \rangle, \\
&\quad \langle Parent = svchost.exe \rangle, \\
&\quad \langle Operation = image\_load \rangle, \\
&\quad \langle Argument = advapi32.dll \rangle \\
&\quad \langle Sequence = 2 \rangle, \\
&\quad \langle Parent = svchost.exe \rangle, \\
&\quad \langle Operation = image\_load \rangle, \\
&\quad \langle Argument = cfgmgr32.dll \rangle \\
&\quad ... \\
&\quad \langle Sequence = 30 \rangle, \\
&\quad \langle Parent = svchost.exe \rangle, \\
&\quad \langle Operation = registry\_modify \rangle, \\
&\quad \langle Argument = /machine/system/controlset001/services/wbiosrvc/... \rangle \rangle
\end{aligned}
$$

The result is a simple description of X that can be easily converted to other threat definition languages or shared directly with others. In our particular case, X represents an anomalous deviation tagged as 'CAPEC-112' by AIDIS. The link to CAPEC provides us with additional semantic information, namely that the corresponding 'Brute Force' label refers to activity where the "attacker attempts to gain access to this asset by using trial-and-error to exhaustively explore all the possible secret values in the hope of finding the secret (or a value that is functionally equivalent) that will unlock the asset." [219]

According to the meta model summarized in Figure 6.7, CAPEC-112 can be an APT kill chain 'Delivery–Intrusion' as well as an 'Installation–Propagation' or 'Installation–Persistence' support action which is typically used in combination with other attack activity. Categorized as the identically named BF (Brute Force) attack action, PenQuest also defines appropriate primary controls countering the threat, namely 'Authentication protection' (AP): This controls group is primarily concerned with managing authenticators such as passwords, tokens, and biometric information. Associated defense actions include the categories 'Remote Access' (REA) and 'Authenticator Management' (AUM), with a range of controls directly out of NIST SP 800-53 [146]. Specific countermeasures therefore include remote access control and encryption, access point management, password/PKI/hardware/biometric authentication, as well as controls related to cache expiration settings.

The information gleaned from the model can now be used to plan appropriate defensive measures to prevent this particular attack. PenQuest's gamified nature also

allows us to play through the attack and test various controls and systems that may reduce threat impact and probability. While the efficacy of the suggested countermeasures need to be evaluated on a case-by-case basis using real world infrastructure or larger-scale expert interviews, the interviewed security practitioners agree that PenQuest is 'applicable to real world scenarios' and that the example mapping makes sense in terms of threat–mitigation pairing.

6.7 Discussion

In this section, we reiterate key features and briefly discuss the implications arising from the evaluation. Future enhancements that go beyond the current implementation are highlighted. In conclusion, we discuss the drawbacks of the current variant of the game and talk about future research.

6.7.1 Features

One of the key aspects of the model is its support for new, hitherto neglected use cases that go beyond the hacking scenario described in this chapter. The general structure of classes presented in Section 6.3.2 is fully compatible with user-side changes to their content. New actors, motivations, enablers and disablers, effects, and attack/defense classes can simply be replaced or amended with low to medium effort, and without any alterations to the core rules. Information derived from external sources (primarily attack patterns and controls) can be changed by applying the introduced data mapping mechanisms – be it semi-automated assignment using intermediate abstraction levels or a natural language approach such as the one sketched in Section 6.5.4.

The network topology introduced in this article – while based on STIX' `VictimTargetingType` TPP schema – remains flexible as well. As long as a attack vectors and dependencies are maintained, game masters/designers can add or remove assets as they see fit. In simulation scenarios it is actually encouraged to model the topology after the real-world system chosen for assessment instead of using PenQuest's default structure. While the process of creating a custom topology is not currently formalized in the rule system, it will be added in the near future to minimize human error.

Assembling new scenarios from the list of actors and assets is a matter of computing all possible actor–goal combinations. The same is significantly harder to do for the game itself due to the combinatorial challenge of pitting each actor, action, enabler, and disabler against one another. It is currently more feasible to use PenQuest to explore an exemplary infrastructure and test various likely attack/defense scenarios in the course of one or several playthroughs. Still, automatically simulating a large number of attack cases is a valid approach to deriving ideal strategies. Such a simulator is one of the major contributions planned for future research and will investigate the utilization of both model checking [59] and reinforcement learning [151].

6.7.2 Limitations

There are a number of limitations that need to be considered before employing the prototype version of PenQuest for attack analysis, risk assessment, or simply as an awareness game. Most of these factors also offer opportunities for future research.

Firstly, one needs to keep in mind that PenQuest currently exists as model with a physical prototype implementation only. While the zero-sum component of system compromise has been realized programmatically (see Section 6.3.3), it is not yet integrated into a fully featured app. This impacts scalability and makes it difficult to effectively reenact large APT campaigns in a sensible amount of time. Ongoing efforts to remedy this limitation include an educational two-player web application expected to be released in 2020. Despite the lack of automation, all modeling and mapping tasks are fully fleshed out and ready for use through cross-referenced tables and the physical game prototype itself. The limited automation and the length of an average game session make the current version of PenQuest best suited for moderated workshops and special lectures.

Another limiting factor pertains the use of external data sources and threat intelligence: Attack patterns may outdate due to a lack of database maintenance and novel threats might not be considered immediately after disclosure. For CAPEC specifically, there are patterns that lack CIA impact or purpose information, e.g. CAPEC-2: "Inducing account lockout". This might make it necessary for the player to expand the repository of attack strategies/actions with their own information.

The assignment of primary controls to defense actions (see Figure 6.8) is currently done semi-manually by a group of IT security experts. Automated mechanisms have proven to be too inaccurate during initial experiments, which utilized a natural language approach using several IT security glossaries. Because of this limitation, the attack–defense mapping of data sources that are significantly larger than the NIST standard might not be feasible. Another minor drawback of the NIST control approach is the small number (19) of orphan controls, which are not related to any other countermeasures. Currently, these controls are not considered in the game, since they would have to be manually added and assigned a specific category, levering out most mechanisms of automation currently in place.

On the CAPEC side, we do not restrict attacks to a specific target type like we do for equipment, since CAPEC does not offer a clean way to assign systems (hosts, network, industrial components, etc.) to a particular pattern. That means that e.g. the BF primary attack such as CAPEC-49: "Password Brute Forcing" can, in the game, be used on an arbitrary victim without further distinction into target categories. This limitation might in some cases simplify a hostile action at the expense of realism. Countering this drawback is possible, but currently requires manual intervention: Depending on the description of the respective attack pattern, the information provided in CAPEC's database (namely the `Summary` and `Example Instances` columns) can be parsed and assigned one of the equipment type categories used for assets ($\langle EffectTarget \rangle$).

Modeling-wise, not all of the 517 CAPEC classes and only a portion of the defensive controls are currently part of PenQuest. Around 12% of the available patterns have been used to populate the model to date, whereas 'detailed' technical patterns referring to specific software attacks (as opposed to 'meta' and 'standard') are not currently included. Control-wise, we prototypically implemented 70 out of 224 controls specified by NIST. With the model itself ready for use, the remainder of the data can be added at will. However, it needs to be stated that the CAPEC repository itself is missing some of the required information needed for automated mapping, which increases the effort required to add the remaining patterns. For future iterations, we will therefore consider alternative vocabularies such as MITRE ATT&CK[1].

Lastly, the limits of the preliminary evaluation itself need to be mentioned: With the relatively small pool of test persons, representative quantitative assessment of the game and model has proven to be difficult. Our current prototype confines experiments to on-site play-testing, and the number of security experts available for evaluating PenQuest's base, game, and rule models who were not involved in the creation of PenQuest is similarly limited – mostly due to geographic constraints and the time required to explain, understand, and assess such a complex model. For this reason, large-scale evaluation has been postponed until the full digital iteration of the game is available, which will eliminate the need for test players to be physically present.

6.8 Summary

In this article, we introduced PenQuest, a meta model designed to present a complete view on information system attacks and their mitigation while providing a tool for both semantic data enrichment and security education. PenQuest simulates time-enabled attacker/defender behavior as part of a dynamic, imperfect information multi-player game that derives significant parts of its ruleset from established information security sources such as STIX, CAPEC, CVE/CWE and NIST SP 800-53. Attack patterns, vulnerabilities, and mitigating controls are mapped to counterpart strategies and concrete actions through practical, data-centric mechanisms. The gamified model considers and defines a wide range of actors, assets, and actions, thereby enabling the assessment of cyber risks while giving technical experts the opportunity to explore specific attack scenarios in the context of an abstracted IT infrastructure.

In summary, PenQuest contributes by:

- Presenting an easily expandable, time-enabled attacker/defender meta model for depicting and assessing advanced persistent threats;
- Developing a set of dynamic, non-cooperative roleplaying game (RPG) rules representing APT campaigns with all their assets, actors, and actions;

[1] https://attack.mitre.org/

- Providing a link between various standards and formats, such as STIX-defined data observables, CAPEC attack patterns, as well as operational risk assessment and mitigation planning within a gamified setting;
- Introducing a mapping mechanism for correlating attacker behavior to opposing security and privacy controls listed in the NIST SP 800-53 standard;
- Presenting and evaluating a physical game prototype ready for deployment in higher education and awareness training;
- Paving the way towards automated attacker and defender strategy inference as well as threat simulation.

We implemented PenQuest as a physical serious game prototype and successfully tested it in a higher education environment. Additional expert interviews helped evaluate the model's applicability to information security scenarios. Key questions asking practitioners if the "abstraction level of attack/defense actions is appropriate", whether the model is "applicable to real-world scenarios", or if the model "appropriately links specific responses to security incidents" were met with a widely favorable response, underlying PenQuest's usefulness for IT threat modeling.

Chapter 7

Star Graph Anomaly Detection and Classification

Contents

7.1 Introduction

In this chapter, we introduce the final Advanced Intrusion Detection and Interpretation System (AIDIS) designed to detect and classify a variety of targeted attack scenarios. Most existing intrusion detection systems do not present the offending behavioral data to the analyst and contribute little to an attack's interpretation. We argue that closing

the resulting semantic gap is a vital next step in holistic IT system threat mitigation. Consequently, our approach contributes by presenting a semantics-aware, fully transparent graph-based anomaly detection system coupled with the automated interpretation of abstracted kernel events collected from Windows workstation computers. AIDIS helps widen the focus of analysis from suspicious binaries to ubiquitous kernel processes that might be affected by adversarial action, thereby enabling uniform anomaly detection for known portions of the operating system. Our approach reduces the computational requirements attached to analyzing each and every application launched on a system and presents anomalies in a human-readable way. Identified outliers are ultimately mapped to our gamified APT meta model, which encompasses widely used threat intelligence languages, attack patterns, as well as mitigating controls. The model can be queried for information and possible responses to past and ongoing attacks while always maintaining the link to the data layer beneath.

In its entirety, AIDIS enables detecting attacks and sharing interpreted threat intelligence for entire OS ecosystems. The accessible combination of attack modeling, white box anomaly detection, and real-world system event data provides a novel approach to combating high-impact threats on IT infrastructures.

In the following, we discuss related approaches and how our system can contribute to current attack detection efforts. Section 7.2 presents related work in the areas of attack modeling and anomaly detection/classification. In Section 7.3, the design considerations of our interpretation system are discussed in detail. AIDIS' technical implementation and all its components are elucidated in Section 7.4, while a full evaluation can be found in Section 7.5. In conclusion, we discuss the current prototype's properties and drawbacks and outline future work.

7.2 Related Work

7.2.1 Attack Modeling

Threat formalisms such as attack/defense models are commonly used by security analysts to share insight, enhance scenario coverage, and help planning and prioritizing threat mitigation by quantifying related system vulnerabilities. Both static and dynamic (behavioral) models are used [248]. AIDIS makes use of 'PenQuest', an advanced dynamic approach realized as a full-fledged strategy game. In the following, we discuss existing work similar to our modeling approach.

AIDIS uses an extended variant of the APT kill chain by Hutchins et al. [133], which represents a simple yet useful solution for modeling targeted attacks by depicting attacker activity as stages in the manner of a tiered military campaign. Several such models have been developed in the past – the decision of which variant to use largely depends on personal preference and data exchange requirements: While the cyber kill chain model [133] considers command and control activity and weaponization as separate stages, the model by Giura and Wang [113] is more detailed when it comes to

the collection of data. Reconnaissance, exploitation, operation, and exfiltration stages are mostly identical, albeit named differently at times. Both models can be used in conjunction with MITRE's APT-enabled Structured Threat Information eXpression (STIX) data exchange format [223], which was developed to represent threat information in a comprehensive manner. In our work, we built upon the general kill chain approach and added sub-stages as well as interdependencies for a more finer-grained view on targeted attacks. See Section 7.3.2 for more information.

Similar to our model in much of its terminology, the Diamond Model of Intrusion Analysis [43] establishes the basic elements of generic intrusion activity, called an *event*, which is composed of four core features: adversary, infrastructure, capability, and victim. It extends events with a confidence score that can be used to track the reliability of the data source or a specific event. While some of its premises are comparable to our own work, the Diamond Model does not consider technical or organizational tools and controls. Mechanisms for determining specific actions conducted on the attacker's or defender's side are not offered. While the Diamond Model is a powerful template in its own regard, our approach aims to provide these mechanisms – and more: PenQuest/AIDIS combine threat modeling with a ready-to-use framework for simulation and automated knowledge discovery. In summary, the Diamond Model and our gamified approach share commonalities and could potentially benefit from each other in terms of feature modeling and terminology.

Syed et al. [303] present a unified cyber security ontology (UCO) extending the Intrusion Detection System ontology by Undercoffer et al. [317]. UCO is a semantic version of STIX [23] with a link to security standards similar to the ones that are used in our work. Real-world knowledge is appended using featured Google searches (Google Knowledge Graph) and various knowledge bases. Syed et al. provide little information about data retrieval mechanisms and general automation. The main use cases emphasized are the identification of similar software and the association of vulnerabilities with certain (classes of) products. Unlike our research, UCO does not consider temporal information or measurements of uncertainty.

Following Schneier's introduction of attack trees [276], the idea of a tree-like representation of an attack scenario, where the root of an attack tree corresponds to an attacker's goal, has been picked up by several other research groups. For instance, Kordy et al. [162] developed a formalism called attack-defense trees (ADTrees), which are node-labeled rooted trees describing not only the measures an attacker might take to attack a system but also the defenses that a victim can employ to protect it. The authors provide semantics and axiomatic definitions that are relevant for further research. However, ADTrees are primarily designed for visualization and require manual mapping of countermeasures – something our model is seeking to remedy. Further works in the area, and at the same time the foundation for Kordy et al.'s work, include attack and protection trees for physical security [87] and attack trees with a temporal component [149].

Many other approaches can be subsumed as taxonomies designed to help analysts or researchers counter specific threats. For example, Mirkovic and Reiher [216] present two taxonomies for classifying attacks and defenses against DDoS attacks. Their approach is to highlight commonalities and important features of attack strategies, a task that is done manually in the light of the problem's complexity. While a common classification scheme is important, our model goes beyond this vital first step and aims to deliver automation support for mapping general attack tasks as described by pattern repositories like CAPEC [219] to standardized defensive controls. This ultimately serves as a foundation for anomaly interpretation and mitigation planning.

7.2.2 Anomaly Detection and Interpretation

Anomaly-based malware or intrusion detection systems are found in many a proposed solution. However, it is rare to see it combined with a semantic component that is dedicated to the automated interpretation of the generated traces, logs, or alerts.

General

The shift of focus towards semantic awareness is visible in several, more general works. For example, Anagnostopoulos et al. [11] present a system for the application of semantics to general intrusion scenarios. The authors seek to classify and predict attacker intentions using a Bayesian classifier paired with a probabilistic inference algorithm. Their semantic model includes both legitimate and illegitimate actors, activities in the form of sequential events, concrete commands issued, and an overall state of attack.

Putting a novel spin on anomaly detection in general, Gautam et al. [112] present a multi-kernel learning approach for One-Class Classification (OCC), an approach where data of only one out of n classes in the dataset is used for training. The authors present an alternative to existing One-Class SVM methods [277] and the Multi Kernel Anomaly Detection (MKAD) algorithm [70] by locally assigning weight to each kernel. While AIDIS relies on binary and multi-class SVM to perform its distinction, the OCC approach will have to be investigated for scenarios that cannot rely on knowledge about classes other than the one defining e.g. benign baseline behavior.

For unsupervised anomaly detection, Zhang et al. [354] introduce a density-based system using the Gaussian kernel function. The objective is to improve outlier detection in non-linear data by assessing the similarity of data points through measuring the local density between a point and a set of its neighbors. In AIDIS, we use three alternative approaches to unsupervised learning for our mostly sequential data, which typically revolve around strings instead of numerals: Grammar inference, heuristic clustering of traces for the generation of template behavior, and text similarity hashing (see Section 7.4.5). Despite the differences in input, Zhang et al.'s take on anomaly detection warrants further investigation as e.g. component in unsupervised trace clustering.

Noble and Cook [236] explore graph-based anomaly detection through the identification of repetitive substructures within graphs as well as by determining which subgraph of interest consists of the highest number of unique substructures and therefore stands out the most. The introduced system is also able to measure the regularity of a graph using conditional entropy. Being mostly formal in nature, the approach does not consider attack semantics. Most other graph-based systems for intrusion detection scenarios discuss attack graphs, which put the focus on (network) vulnerability analysis and the sequence of events leading to a state of compromise [247, 289].

Host Activity

Kruegel et al. [166] describe a classical approach to detecting anomalies in call sequences. Their system is designed to detect attacks against privileged applications. To this end, it analyzes the relation between system call arguments and calling contexts. Among the anomalies considered are string length, character distribution, as well as the occurrence of certain characters. In contrast, AIDIS mainly considers file system activity linked to a central graph node representing the calling process.

Dolgikh et al. [79] conduct dynamic behavioral analysis of applications. Their system is capable of automatically creating application profiles for both malicious and benign samples. It considers recorded API calls that are subsequently transformed into a labeled graph representing a stream of system calls. Graphs are compressed using a genetic data processing algorithm in order to extract behavioral profiles. There is no classification interpretation or interpretation of the resulting data.

Anderson et al. [12] present a detection algorithm based on the analysis of graphs constructed from dynamically collected instruction traces. Working with simplified assembly sequences represented as Markov chain and subsequently transformed into a weighted directed graph, the general level of abstraction is lower than in AIDIS, which primarily uses generalized API calls. For classification, Anderson et al. [12] use SVM on previously created similarity matrices. The system's specific results are further discussed in Section 7.5.4, where they are compared to AIDIS' binary classifier.

Touching the network domain, Jacob et al. [141] present Jackstraws, a system designed to identify command and control (C2) communication. Unlike other network-centric approaches, Jackstraws uses host events captured through dynamic analysis. Association of network activity to local processes is achieved through behavior graph modeling of data flows between individual system calls. Graph templates for C2 pattern similarity matching are mined from a known set based on a technique introduced by Yan and Han [350], followed by a clustering stage. While Jackstraws is potentially vulnerable to certain obfuscation and mimicry attacks [329], its unique approach to detecting C2 traffic makes it interesting for both APT detection and knowledge generation. Jacob et al. [141]'s work is further discussed as part of the comparative evaluation in Section 7.5.4.

Network Domain

On the network traffic side, Münz and Carle [232] present TOPAS, a traffic flow and packet analysis system compatible with NetFlow and IPFIX. The system's algorithm uses threshold-based detection via pre-defined values as well as outlier detection through the comparison of a sample to previously learned, nominal behavior. While this offers a good foundation for traffic anomaly detection, the link to local processes and applications is not investigated.

The system presented by Ambwani [10] focuses on full network traffic dumps associated to DoS, privilege escalation, and other remote threats. It is similar to AIDIS in its use of SVM multi-class classification for the identification of various attacks, underlining the feasibility of such ML approaches for classification tasks. Anomaly detection or threat modeling is not part of the system. We compare the system's accuracy to AIDIS in the evaluation in Section 7.5.4.

The work by Vance [322] is one of the few approaches that focus directly on APTs: He describes a flow-based monitoring system that uses statistical analysis of captured network traffic data to detect anomalies – instead of event-based deviations. Vance's system uses change detection to identify flows that hint at command & control traffic, data mining, or exfiltration activities.

As above works exemplify, none of the solutions quite manage to bridge the gap between system events (be they function calls or traffic flows) and a truly meaningful representation of an attack in its entirety. Closing this semantic gap is one of the main goals of the system presented in this work. For more related work in the domain of semantics-aware APT detection, refer to Chapter 2.

7.3 System Design and Model

Our proposed system revolves around the hypothesis that the observation of ubiquitous OS kernel processes is a feasible alternative to sample-focused analysis, where extracted binaries are checked for malicious properties and behavior. Unlike suspicious samples, system processes are present at all times and do not need to be identified prior to analysis. However, it is not entirely clear which processes deserve our scrutiny the most – something that AIDIS is built to address through its highlighting of expressive system applications.

Furthermore, we hypothesize that classifying identified anomalies is preferable to classifying entire traces of unknown behavior. Instead of processing all captured events associated to a process or a full user session, the system identifies and extracts anomalous events prior to classification. This reduces the amount of data necessary for interpretation and provides the analyst with not only a pattern of potentially harmful activity, but also a more accurate verdict as to which class/stage of attack it may belong to.

Since the computational complexity of any graph-based solution is likely to be high, any such anomaly detection and explication system will have to consider means to reduce processing times. With AIDIS, we opted to utilize compression through grammar inference, which transparently creates rules from sequences of events that are synonymous to nominal behavior – in contradistinction to uncompressed terminals representing outliers. By replacing commonplace activity with rules, we can save both time and computing resources, while at the same time offering advanced threat formalization capabilities.

Lastly, we hypothesize that the interpretation of previously disseminated anomaly data (i.e. events that constitute a deviation) by mapping it to a dedicated attacker/defender model can, in conjunction with automated anomaly classification, be used to explain threats in a comprehensible manner. For this reason, we map all results to PenQuest, the model component of AIDIS, which provides a link to our enhanced APT kill chain as well as various threat vocabularies.

In summary, the proposed system provides the following functionality:

1. Identification of relevant OS kernel processes for continuous, sample-independent monitoring;
2. Compression of event data through grammar inference;
3. Automated detection and classification of anomalies;
4. Mapping of classified anomalies to a model for threat explication and reasoning.

Before a technical implementation can be approached (see Section 7.4), we have to consider several formal, semantic, and strategic factors. In the following, we discuss these initial premises by following a design checklist (see Table 7.1) developed for the assembly of semantics-aware systems countering advanced threats to information systems.

7.3.1 Design Checklist

The design of the system, which has been independently discussed in [189], is based on the roadmap for a conceptual APT defense system introduced in a survey by Luh et al. [187]. In order to fulfill the requirements for the comprehensive detection and analysis of targeted attacks we followed the presented checklist and extended the design with the ability to explain detected anomalies in behavioral data through a combination of classification and threat modeling.

Referencing the design objectives for semantics-aware detection and analysis systems (Chapter 2), AIDIS fulfills the suggestions for the scope of implementation–and more. Table 7.1 provides an overview. Specifically, our approach considers both OS and network events, fuses threat detection as well as attack analysis into a behavior-based anomaly detection system able to classify, extract, and interpret hostile activity, and combines threat intelligence and response into a common model. The system's detection methods and analysis techniques encompass anomaly detection on behavioral data,

	Prediction	Prevention	Intelligence	Correlation	Detection	Analysis	Response
G met [5/7]		✓	✓	✓	✓	✓	
	Malware	Host intr.	Network intr.		Attribute	Behavior	Context
T countered [3/3]	✓	✓	✓	A used [3/3]	✓	✓	✓
K for T [3/3]	✓	✓	✓	G for A [3/3]	✓	✓	✓
Correlation for T [3/3]	✓	✓	✓	Corr.: A [3/3]	✓	✓	✓
	Threat info	System logs	App logs	Network events	Local events	Binary/raw	Alerts
I incorporated [3/7]	✓			✓	✓		
	Recon	Weaponiz.	Delivery	Exploitation	Installation	C2	Actions
APT stage covered [6/7]	✓		✓	✓	✓	✓	✓

Table 7.1: Design checklist after Luh et al. [187]. Capabilities and properties of AIDIS are highlighted. Prevention through risk assessment and awareness building is provided by the PenQuest meta model (see Section 7.3.3) that ties the technical components (intelligence, correlation, detection, analysis) together. While the resulting anomaly reports and scores enable a defender's response, the mitigation of the attack itself is not part of AIDIS. Input data includes kernel events describing local and network events; threat information is brought into the mix by the underlying model. Abbreviations: G...goal, T...threat type, A...analysis technique, K...knowledge generation, I...input data type.

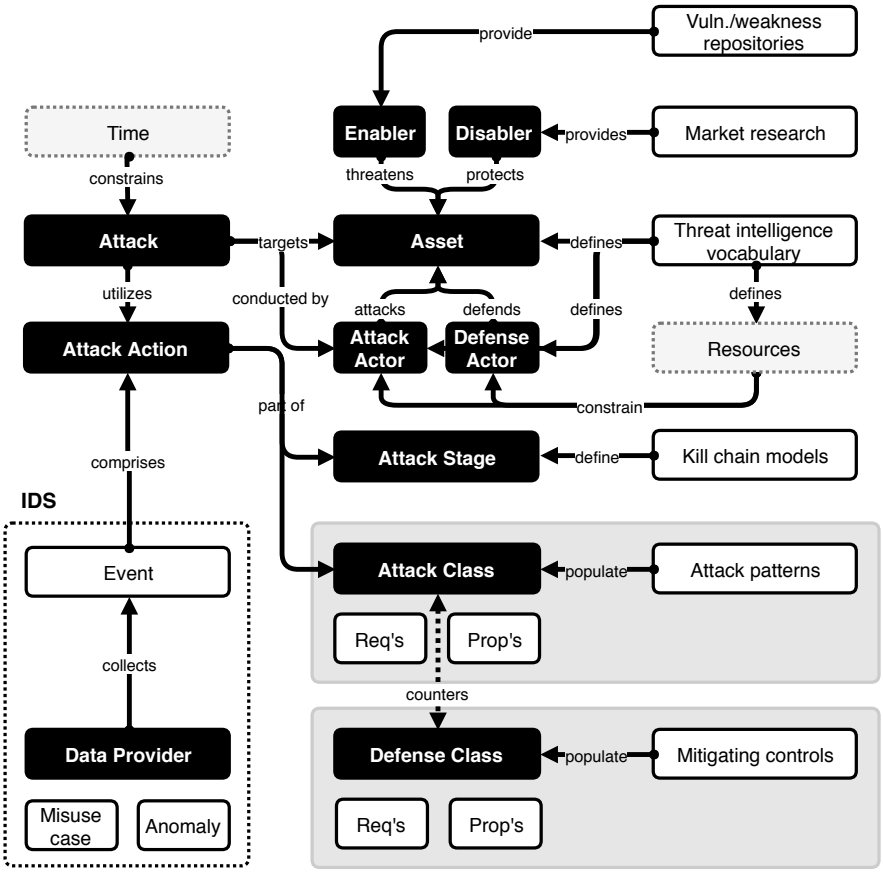


Figure 7.1: Simplified representation of the PenQuest meta model. The lower left side depicts the AIDIS data provider (agent) monitoring for anomalies or pattern occurrence, while the right sight shows PenQuest's class structure for a generalized Action X.

threat context, classification, and reusable (attack) patterns extracted from the original corpus. These unique combination of features helps to make AIDIS an expedient response to the increasing complexity of targeted attacks.

In the following subsections, we talk about the general threat definition as well as the underlying attack/defense model, leading up to the technical implementation.

7.3.2 Threat Definition

For a definition of high-level threat stages we decided to extend the cyber kill chain model by Hutchins et al. [133]: Every APT stage is further split into subcategories (see Figure 6.7) that are ultimately linked to concrete attack actions provided by the underlying attacker/defender model. For APTs, attack stages typically have successor stages that may be executed once its predecessor has been successfully completed. For example, Exploitation attacks require the prior completion of a Delivery action, lest the utilized malicious code cannot be executed on the target. The APT kill chain categories used to define the general threat are discussed in detail in Section 6.4.5.

The adapted APT kill chain is only one part of the model underlying AIDIS. Refer to Chapter 6 for general information on the gamified component of PenQuest and how it can be used to explain, simulate, and help mitigate a threat.

7.3.3 Attack Modeling

For attack modeling and the subsequent interpretation of classified system behavior, we utilize PenQuest, our versatile attacker–defender meta model that takes the definition of threat stages and provides concrete actions based on accepted security languages and standards. See Figure 7.1 for an overview of the model. Refer to Chapter 6 for detailed information about the model, its gamified rule set, and specific action definitions.

7.4 Core Components

AIDIS is composed of several components enabling the underlying anomaly detection and knowledge explication process. The initial tasks encompass the acquisition of data on a number of monitored machines and/or network devices as well as the transmission and translation of kernel events to a clean database format. In stage 2, we extract OS processes suitable for observation through sentiment analysis. Following optional data compression using grammar inference, we link events by their contextual parent and construct traces in the form of star structures [353], simple graphs that describe the operations conducted by each process within a specific time range. From a baseline of benign system behavior we then extract one or several process-unique templates that are subsequently used to check new activity for anomalies by measuring the edit distance between the simplified graphs.

Our approach not only calculates deviations but also returns a human-readable list of actions that constitute the identified anomaly. This list is ultimately classified using both a Random Forest and SVM-based approach. The resulting behavioral patterns are mapped to the aforementioned PenQuest model, thereby linking each anomalous action to an APT attack stage and semantic description. Figure 7.2 and Table 7.2 provide a full overview of the system components. The following subsections detail each stage and provide technical specifics.

7.4.1 Data Collection

AIDIS works with *events* collected directly on the host (endpoint). Event traces (i.e. ordered lists) are typically defined as descriptions of operating system kernel behavior invoked by applications and, by extension, a legitimate or illegitimate user. Individual events are abstractions of raw system or API calls that yield information about the general behavior of a sample [330]. API calls may include wrapper functions (e.g. `CreateProcess`) that offer a simple interface to the application programmer, or native

Component	Sentiment analysis	Grammar inference	Graph anomaly detection	Anomaly classification
Chapter/Section	7.4.3	7.4.4	7.4.5	7.4.5
Optional stage	Yes	Yes	No	No
Algorithm(s)	LLR [84, 190]	Sequitur [234, 193]	Kuhn-Munkres [131]	RF [181], SVM [64]
Knowledge generation	✓✓	✓✓	✓	✓
Anomaly detection	✓✓	✓✓	✓✓	✗
Classification	2 classes	✓	2 classes	n classes
Interpretation	✗	✓	✗	Yes
Learning	Supervised	Unsupervised	Supervised	✗
Complexity	$O(n)$	$O(n)$	$O(n^3)$	$O(k * nlog(n))$, $O(n^2)$

Table 7.2: List of AIDIS components beyond collection and preprocessing. Knowledge generation describes the extraction/inference of information about event sequences and anomalies in general. Anomaly detection capabilities allow for the identification of anomalous behavior through a score, statistical analysis, or visual assessment. Classification enables the separation of the result into malicious, benign, or more granular threat categories, while further (anomaly) interpretation is enabled through the link to our PenQuest model. Different learning modes are identified, as are the component's computational requirements. Legend: ✓✓...fully supported/key feature, ✓...limited support/byproduct, ✗...not supported or not part of the primary purpose.

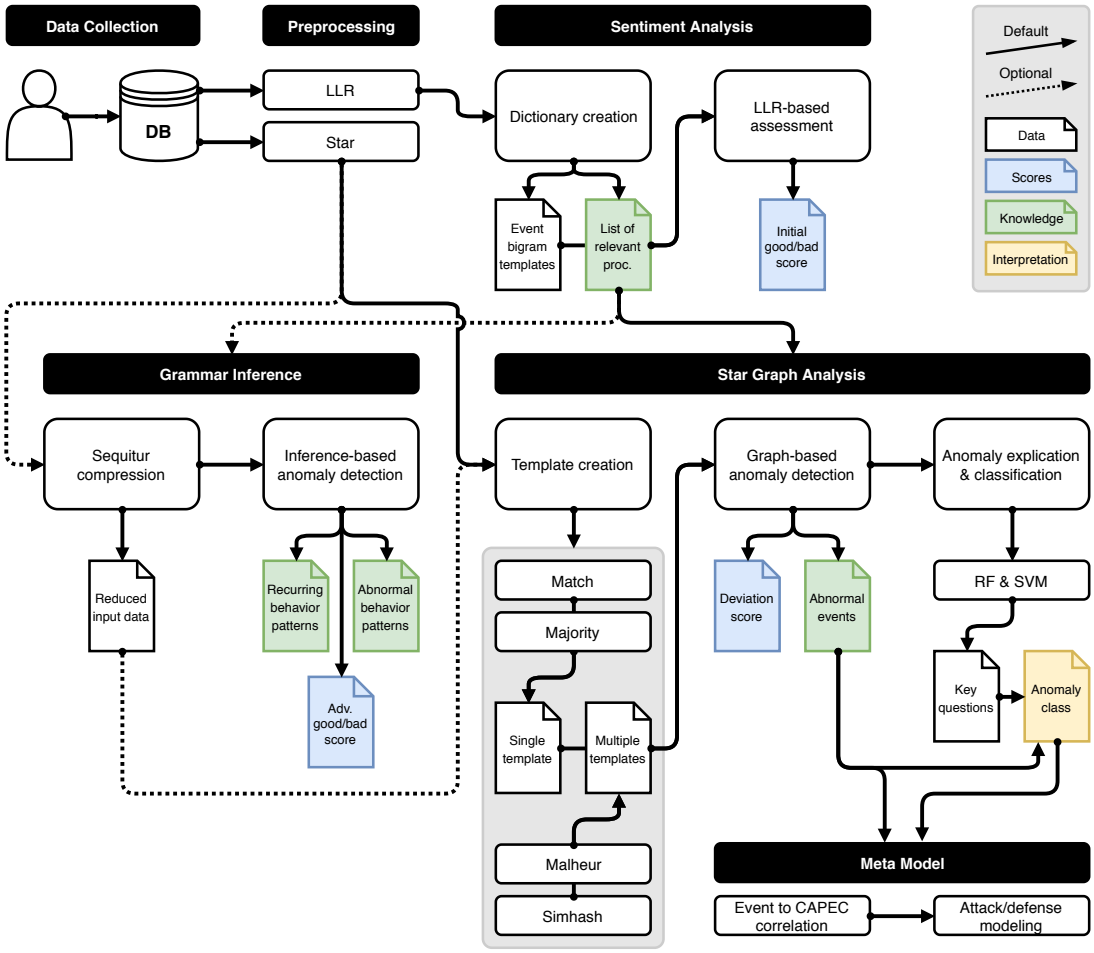


Figure 7.2: AIDIS system overview. Optional sentiment analysis is used for extracting kernel processes that deserve special attention, while the grammar inference component offers data reduction and unsupervised anomaly detection. The core knowledge extraction and anomaly detection component utilizes star structures for template generation and matching. Event interpretation is realized through RF and SVM classification applied to the resulting anomaly reports. The link to our meta model (see Figure 7.1) semantically enriches the information and helps plan an appropriate response.

functions (e.g. `NtCreateProcess`) that represent the underlying OS or kernel support tools. In its abstracted form, a contextual event trace might look like this:

ProcessEvent	Start: ‘shell.exe’ (PID 220)
ImageLoadEvent	Load: ‘library.dll’
RegistryEvent	Open: ‘HKLM/Software/.../Run’
RegistryEvent	Add: REG_SZ (‘evil.exe’)
FileEvent	Create: ‘evil.exe’
ProcessEvent	Start: ‘evil.exe’ (PID 224)
ProcessEvent	Terminate: ‘shell.exe’ (PID 220)

Table 7.3: Example trace of events as chronological list.

AIDIS collects process and network event data directly from the Windows kernel. We employ a driver-based monitoring agent designed to collect and forward a wide

range of events to a database server. This gives us unimpeded and fast access to events depicting various OS operations [201]:

- **Process events** – Whenever a process is started or stopped, the monitoring system registers a new event. Next to PID and paths, we record parent and contextual information such as ownership data. Process events are at the heart of our system: Every other type of event is ultimately associated to a process through its PID and timestamp (see below for more information).
- **Thread events** – Some events are triggered by individual threads instead of processes. The information logged by the agent is largely similar to process events; the main identifier for threads is the thread ID (TID).
- **Image load events** – Most processes load additional resources (functions) stored in various program libraries (DLLs). The nature of a DLL can give a good indication as to which behavior the binary executable will exhibit during its lifetime.
- **File events** – File events are logged when a file is read, created, accessed, modified, or deleted. Logging file interaction is important since processes can interact with virtually every file stored on the disk. Attack-related file events can e.g. help identify dropped executables or data theft.
- **Registry events** – Applications use the registry to save user and program settings while other hives contain startup programs or file type settings. Since it is a common target for espionage and system manipulation attacks, monitoring registry events is critical for any Windows-based detection solution.
- **Network events** – Network events encompass the handling of inbound and outbound connections as well as the access to general OS networking resources. Depending on the nature of the process, network events can be used as indicator for malicious behavior, since many malware variants contact remote systems for e.g. command & control purposes.

The relative ease of monitoring as well as the semantic expressiveness of kernel events and network operations make such traces ideal for dynamic software and, by extension, malware analysis as well as application classification. The system introduced in this thesis uses this rich repository of behavioral data to compile sentiment dictionaries as well as graph-like star structures of event sequences that can describe not only a single application, but a system session in its entirety. This approach is detailed in the following subsections.

7.4.2 Preprocessing

All previously collected events are linked through their parent process in order to establish a semantic connection between action and cause. This is realized through two attributes that are present in all the data collected by the host monitoring agent: Creation time, and the PID that forms a unique identifier for each process. Threads work in a similar fashion. Like PIDs, thread IDs (TIDs) are appended to other event types

(e.g. registry events). Ultimately, each process or thread created by the respective event can incorporate an arbitrary number of child events, depending on its nature and run time.

Both process and thread events can be used to construct an event tree depicting the flow of file system activity that helps to determine specific dependencies between processes and general events. Concatenated into a full system graph, the sequence of events constituting a monitored session are assembled without orphan entries (depicted in Figure 3.3) interrupting the process flow by grouping them by their associated process and thread. These pseudo-chronological *smart traces* (see Chapter 4 and [201]) are the basis for all follow-up computation.

Further preprocessing includes the normalization of non-uniform IDs such as user names, security identifiers, and temporary folder names. This is done to make data more comparable across systems and to prepare the traces for future anonymization.

7.4.3 Sentiment Analysis

AIDIS uses an approach akin to sentiment analysis [111] for generating initial knowledge about relevant OS processes. In this optional stage, we determine the most expressive process candidates for later investigation. At the same time, this kick-off stage computes a first benign/malicious score that provides us with a tendency towards general harmfulness for the provided dataset. The resulting verdict can be used as additional feature in the final classification stage of the AIDIS process. Figure 7.2 and Table 7.2 show how this component fits into the big picture of AIDIS.

The details of the sentiment analysis component have been previously discussed as stand-alone anomaly detection solution based on inferred dictionaries containing 'malicious' vocabulary. While the evaluation found in this chapter instead focuses on the selection of relevant processes for continuous monitoring (see Section 7.5.3), the technical foundations of the core component remain the same. Please refer to Chapter 4 for more information.

7.4.4 Grammar Inference

Grammar inference through Sequitur compression is the second optional stage in the AIDIS process. It is used to losslessly reduce the amount of input data for the more computationally expensive final stages, while providing a semi-supervised approach to identifying potentially interesting portions in arbitrary event sequences and smart traces.

For the purpose of compression we utilize SEQUIN (Chapter 5), a grammar inference system based on the Sequitur algorithm, which constructs a context-free grammar (CFG) from string-based input data. Specifically, Sequitur is a greedy compression algorithm that creates a hierarchical structure from a sequence of discrete symbols by recursively replacing repeated phrases with a grammatical rule [234]. The algorithm creates this representation through two essential properties, which are called *rule utility*

and *bigram uniqueness*. Rule utility checks if a rule occurs at least twice in the grammar, while bigram uniqueness observes if a bigram occurs only once. A bigram in this context describes two adjacent symbols or terms.

The full rule extraction and evaluation process is detailed in Chapter 5. There we describe the application of our adapted Sequitur system on smart traces of kernel events associated with arbitrary processes and other security-relevant data, proving a full example grammar. In short, SEQUIN has a wide variety of applications that go beyond AIDIS: Table 5.3 provides an overview.

The reduction of input data in particular can be helpful to decrease the complexity of lengthier analysis tasks, such as the graph-based approach discussed in this chapter (see Section 7.4.5): By using SEQUIN, it is possible to slim down the input corpus to only relevant n-grams ($n \geqslant 2$), instead of working with the full, unfiltered set of event or code snippet unigrams. SEQUIN's rule transformation mechanism (5.4.4) also enables us to work with an automatically generated placeholder variable instead of several compound terminals.

See Section 7.5 for an evaluation of the compression component in the context of AIDIS. Refer to Chapter 5 for more information on SEQUIN.

7.4.5 Star Graph Analysis

Whether or not relevant processes have been identified using sentiment analysis and input data has been compressed, the key analysis component of AIDIS can be executed at this point: We utilize *star structures* to create a by-process representation of event sequences that encompass single process launch behavior, its full run time, or even entire multi-process system sessions. Star structures are a means to reduce the complexity of a known NP problem to polynomial complexity [131, 353]. Instead of searching entire system session graphs for matching patterns, the star structure approach breaks down the computation into a triplet of nodes (vertices) connected by a labeled edge, denoted as $G = (U, V, E)$, where U and V are nodes and E is the respective edge. The attached label is used as basis for minimal cost calculation of same-size star structures. Specifically, we utilize bipartite graph matching based on the Hungarian (Kuhn-Munkres) algorithm [167], where every star is processed as a matrix. Graph edit distance calculation determines the minimal costs of relabeling the nodes and edges of a graph G to match a target graph H. The edit path $P_{G,H}$ can be understood as a sequence of transformation operations σ. The final graph edit distance is determined by the cheapest of all edit paths between G and H.

Compared to full graph matching, this approach is typically considered to be a faster, but less precise approximation, as it only matches the immediate neighborhood of one node at a time. In our system, we use an adaptation of the Hu et al. [131] approach that combines n bipartite graphs into one star representing a single process. This makes the effect on result accuracy far less pronounced: With a focus on individual processes, our input data can already be reduced to star structures without significantly

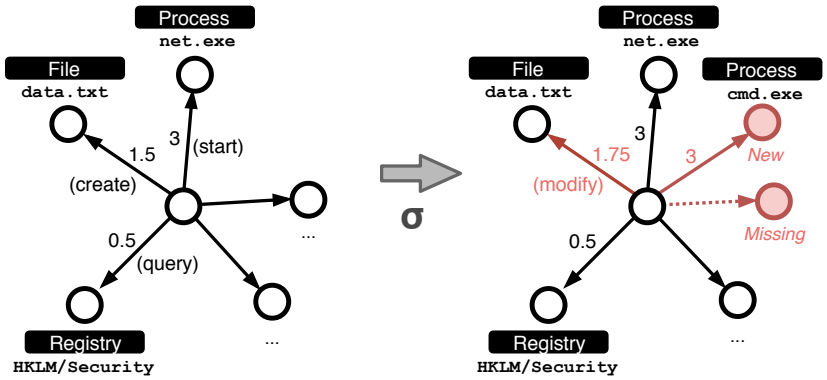


Figure 7.3: Example event representation for process `svchost.exe` (central node). Target graph H (right) differs from the baseline graph G (left) by several additional or missing events, depicted as red nodes. Mere changes to the edge label (different operation type applied to the same object) are considered as well. Graph transformation σ is derived using the Kuhn-Munkres algorithm [167].

compromising trace semantics, because we anchor every event to a trigger (parent) process (see Section 7.4.2) that actively invokes respective actions, making this process the natural center vertex of a star-shaped graph. In our system, elemental operations for determining the minimal cost graph edit distance between individual elements are not limited to relabeling nodes, but consider the connecting edges as well. There are two operations that contribute to the edit score:

Vertex edit operations encompass single vertex relabeling σ_{RV} as well as both an insert vertex σ_{IV} and a delete vertex operation σ_{DV}. Semantically, each vertex is akin to an unspecified system event (event type plus parameter, sans event type) as introduced in Section 7.4.1. Depending on the type of operation, the respective event in H is either new (insert), missing (delete), or has been altered (relabel) from the baseline G.

Edge edit operations, on the other hand, primarily consider the edge relabeling cost σ_{RE}. We opted to dynamically assign individual relabel costs based on the type of event considered, making the approach fully capable of assessing event similarities. For example, σ_{*V} will drastically increase in cost when e.g. converting a semantically inexpensive file 'read' event to a relatively high-impact 'delete' event. The type of operation (numerical representations of e.g. create, modify, delete, start, and stop operations) considered by σ_{RE} determines the final cost of edge relabeling. See Table 7.5 for a list of event types and their experimental labels. Combined with a vertex operation, all possible changes to a process can be quantified.

Figure 7.3 shows a simplified example. In the depicted case, the base graph consists of various vertices representing events such as the creation of a file, the start of a process and an open/read operation conducted in the `HKLM/Security` hive of the Windows registry. When comparing the baseline graph G to a target H, the introduced Hungarian graph edit distance approach will use σ to determine the minimal cost of transforming G to H. In case of the exemplary file event interacting with `data.txt`, this edit distance

Method	Perfect match	Majority	Prototype	Sim. hashing
Algorithm(s)	String comp.	String comp.	Malheur [263, 314]	MinHash [40] Jaccard sim. [138]
Deterministic	✓	✓	✓	✗
Template count	1	1	n	n
Reduction	✗	✗	✗	✓
Complexity (extraction)	$O(n)$	$O(n)$	$O(k * n)$	$O(n^2)$
Complexity (matching)	$O(n)$	$O(n)$	$O(k * n)$	$O(k * n)$

Table 7.4: Overview of star graph template creation methods. Single template approaches are well suited for simple processes with little semantic variance. Multiple templates are needed for complex, multifaceted processes. Similarity hashing is the only method that supports the reduction of Malheur-derived templates but is less accurate when used for extraction due to its non-deterministic nature. It is computationally advantageous to reduce Malheur templates using similarity hashing.

is a mere 0.25, since a single σ_{RE} operation is sufficient to transform the bipartite graph $G(svchost.exe, 1.5, file.txt)$ to $H(svchost.exe, 1.75, file.txt)$.

This method for determining the minimal edit distance between two star-shaped graphs is used as the foundation for context-aware anomaly detection utilizing supervised learning on a per-process basis.

Star Anomaly Detection

The required transformation operations and, by extension, the minimal graph edit distance between two star structures can be used to determine the event-level deviation between instances of the same process. In order to automatically determine thresholds for each observed process, we first need to create a template from a benign environment. Only then can we match base to target graphs and disseminate their differences.

We have implemented the generation of baseline templates in 4 different ways (also see Table 7.4), two of which generate a single template, while the other two produce any number of templates ranging from 2 to n, depending on the complexity and versatility of the process/session in question. In each case we take a set of benign process graphs and extract an optimal representative using one of the below methods:

Perfect match – Here, we extract identical events found in each iteration of a process (i.a. the 'smallest common denominator') and assemble an entirely new graph. This creates a sleek template that enables the analyst to primarily focus on hitherto unobserved events. However, there is an performance-for-accuracy trade-off that results in higher mean edit distance values. The approach proved to be best suited for background processes with a single purpose and little user interaction.

Use-case examples: Quick Access for Intel Grapics (`igfxtray.exe`), Office Telemetry Agent (`msoia.exe`), Windows Power Management (`powercfg.exe`).

Majority – This method picks the most common base graph from the input set and converts it to a template without altering its contents. While slightly more accurate than 'perfect match', this approach struggles with processes that show greater variety in their benign instances due to their multifaceted nature.

Majority extraction is generally similar to the perfect match approach. It is best utilized for processes where major deviations from the baseline are not tolerated. Examples would include applications that exhibit behavior following a set schedule or isolated processes with little system interaction. Despite its limitations, majority matching is still the best choice for complex processes where a multi-template approach is not desired for e.g. performance reasons. Certain concessions regarding accuracy have to made, however, as is discussed in the Evaluation chapter below.

Use case examples (in addition to the above): Chrome Updater (`GoogleUpdater.exe`), Windows Activation Client (`slui.exe`), Java Launcher (`java.exe`).

Prototype extraction – Especially useful for diverse processes, this approach uses the Malheur algorithm [263, 314] to extract not one, but several prototypes representative of the various aspects of a single process. This promises significantly improved accuracy when assessing more complex OS applications. However, the resulting number of templates sometimes negatively impacts performance, which is why we added a second component that intelligently merges templates using similarity hashing.

Prototype extraction is best used for complex processes which control a wide range of OS functions and that are not necessarily similar in their behavior. In a best-case scenario, each functionality is automatically assigned a template. If newly logged behavior does not correspond to at least one of them, the observed activity is treated as an anomaly.

Use case examples: Windows Generic Host Process (`svchost.exe`), Generic Host Process for Libraries (`taskhost.exe`), Registry Editor (`regedit.exe`).

Similarity hashing – Usable as both standalone alternative for the Malheur approach as well as a reduction mechanism for the same, this take on multi-template creation is based on the MinHash algorithm [40], which builds upon the mathematical concepts of resemblance and containment to measure document similarity. Specifically, we measure the differences between original traces or previously Malheur-extracted templates by their Jaccard distance [138]:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}, where(A,B) \subset U$$

In short, the MinHash algorithm converts a set of tokens from U into n randomly selected and hashed tokens, which are then broken down into bands and compared [252]. Documents are considered similar if the resulting Jaccard distance threshold is exceeded. In AIDIS, the individual document similarity values are mapped to a graph, where representative prototypes are determined by their betweenness centrality (see Figure 7.4).

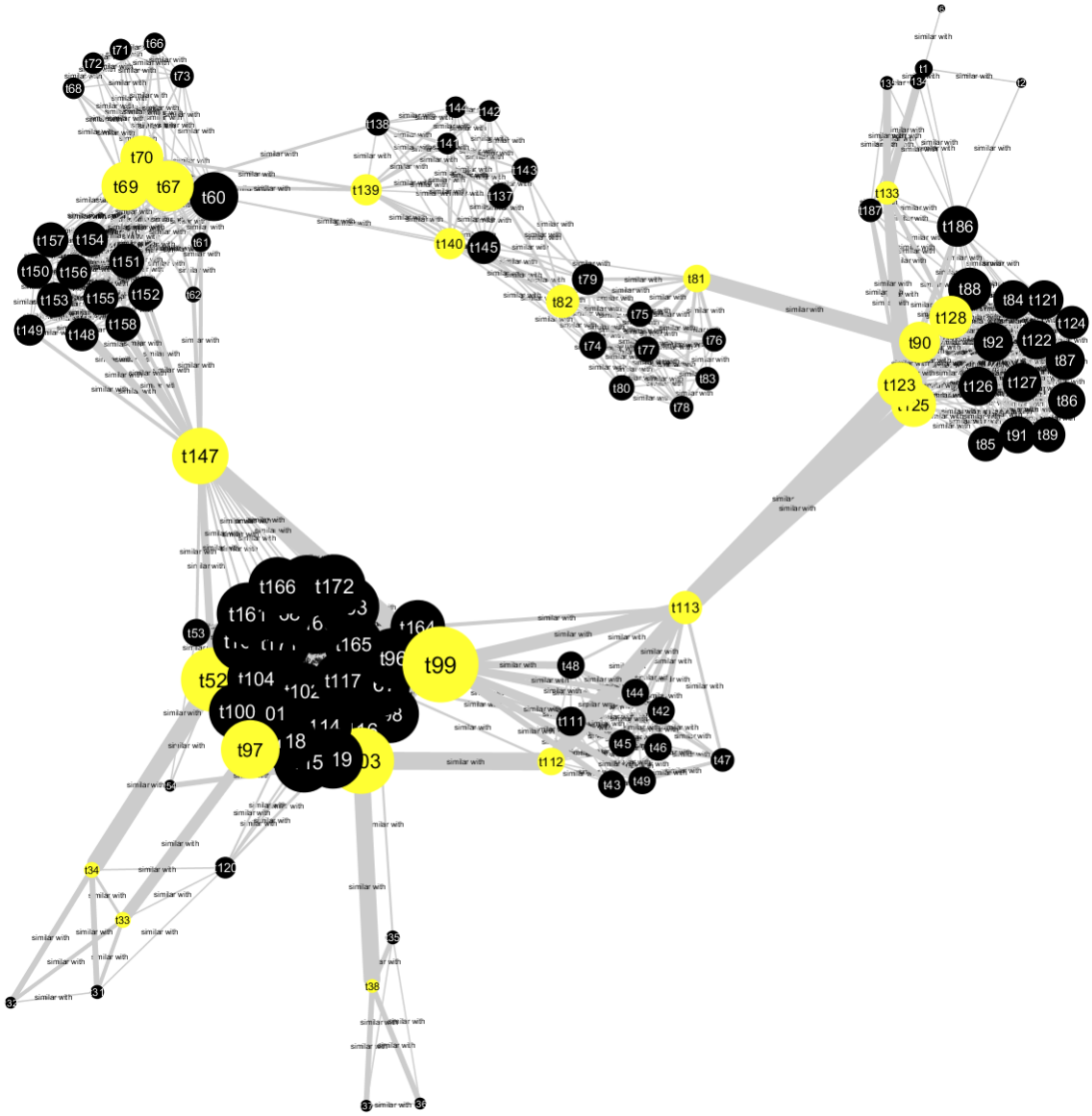


Figure 7.4: Example template extraction through similarity hashing. Similar templates are determined by their Jaccard distance and graph betweenness centrality. The yellow nodes were determined to have the greatest betweenness centrality score. All matches with a Pareto score of $\geqslant 90\%$ are ultimately chosen as templates.

If betweenness centrality is equal, node in-degree is used instead. The most significant traces, determined by a Pareto score of $\geqslant 90\%$, are ultimately kept as templates.

Similarity hashing has proven to be a feasible second stage to the prototype extraction approach. For complex processes, it reduces the number of Malheur prototypes to a more workable number without negatively impacting accuracy. Additionally, similarity hashing is well suited to processes that are versatile in nature but similar in their behavior.

Use case examples: File Ownership Tool (`takeown.exe`), Task Scheduler (`schtasks.exe`), Session Manager (`smss.exe`).

Before any anomaly detection can be implemented, we need to set threat thresholds. In our case, they are determined by comparing the generated template(s) to the

remainder of the benign input graphs using the Hungarian distance. This yields a minimum, mean, median, and maximum lower-bound edit distance for each process. Depending on the level of scrutiny, each of these distances can be used as anomaly threshold. In our case, the default (mean) threshold was dynamically derived from a high number of test runs.

Armed with one or several templates for each process, we can now check unknown graphs against the predetermined thresholds and extract events responsible for the deviation.

Star Anomaly Classification

As our system focuses on the classification of anomalies instead of unknown system traces in their entirety, the amount of data processed in this explication stage is drastically reduced. Specifically, we seek to explain why the anomaly detection routine has identified a star structure as significantly deviating from the template, thereby disseminating the in-depth knowledge gained in the process. Only afterwards can we commence with anomaly classification.

Knowledge dissemination – One of the advantages of our anomaly detection system lies in the fact that the star depiction of a graph allows for the comprehensive dissemination of semantic information that depicts each and every anomaly in a simple fashion. The analyst is presented a report detailing the events that constitute the respective deviation. Below snippet shows an example `svchost.exe` process being checked against one of its extracted prototype templates:

```
==> svchost.exe [Deviation (threshold): 300.5 (123.3) --> ANOMALY]
svchost.exe spawns 13 additional threads
svchost.exe terminates 18 additional threads
svchost.exe loads 54 additional images
=> atl.dll
=> bcrypt.dll
   (...)
svchost.exe sets 6 additional registry entries
=> /HKLM/Software/Microsoft/...
   (...)
svchost.exe modifies/deletes 6 additional files
=> /Windows/system32/acctres.dll
   (...)
svchost.exe opens 7 additional network sockets
=> 192.168.100.100
   (...)
```

As mentioned in Section 7.4.2, AIDIS allows for varying levels of granularity. Registry paths can either be normalized to hive names or be processed in their entirety. Abstraction of IDs, memory addresses, user IDs is implemented as well – as is pseudonymization of file names, IP addresses, and other personally identifiable information.

Knowledge interpretation – Knowledge dissemination offers interesting information to the analyst but does not yet automate its interpretation. One of the key components of our system is the classification of certain combinations of anomalous events by mapping them to both APT attack stages contained in the model (discussed in Section 7.3.2), as well as CAPEC attack patterns describing even more concrete adversary behavior (Section 7.3.3).

The initial version of our system explores event combinations using Random Forest and linear kernel Support Vector Machines (SVM). In the first step, we use the disseminated knowledge to answer over 200 competency questions that are expected to aid in the decision of whether a factor contributes to a malicious objective of a certain kind. These questions include simple Boolean queries into the presence of events over another event (e.g. if the number of thread terminations exceed the number of thread spawns) as well as decisions based on the presence of certain activity tags describing the base functionality of a loaded image (e.g. networking, authentication, user interface, kernel, etc.). The latter is enabled by intelligent tagging (categorization) of more than 1,700 known Windows function libraries, which has been done in advance by parsing both the Windows API section of the MSDN library[1] as well as community sources[2]. Pattern checks determining the use of certain system directories for file events or the assessment of IP addresses are technically possible, but were not implemented at this point: The goal of the prototype system was to avoid fixed patterns as much as possible, as they require constant tuning effort and might be circumvented by a malware's analysis evasion routines.

In order to support the development of expressive competency queries we apply the Random Forest algorithm to determine the mean decrease in accuracy/Gini for each feature, thereby selecting the most significant questions for the respective scenario. For the discrimination of anomaly traces, both binary 'benign' vs. 'malicious' and multi-class classification is used: Based on the response to the competency questions, we get a probability describing the graph's affinity towards a certain kill chain stage or attack pattern. The process was double-checked using a linear kernel SVM with and without hyperplane optimization. See Section 7.5 for the list of assigned classes as well as detailed evaluation results.

Ultimately, AIDIS maps the resulting verdict (e.g. 'anomaly belongs to class CAPEC-112') and the anomaly report itself to our PenQuest model (Chapter 6), thereby building our knowledge base of labeled attacks that can then be associated a goal, stage, likely actor, possible countermeasure, and more. From then on, the data would be seen as part of the model and can be viewed in context, presenting analysts with classification and interpretation of hitherto unknown events. See 7.5.3 for an example mapping based on real-world data and Section 6.5 for a closer look on the modeling aspect.

[1] https://msdn.microsoft.com/en-us/library/
[2] https://undocumented.ntinternals.net/

7.5 Evaluation

In this section we discuss the experimental setup as well as the individual steps of the AIDIS system, beginning with the identification of relevant processes through LLR sentiment analysis. Figure 7.2 and Table 7.2 provide an overview of how the individual components play together. In summary, AIDIS provides the following: 1) Data collection, 2) tree and trace construction with data cleanup, 3) LLR-based sentiment analysis for relevant process identification, 4) optional compression through grammar inference, 5) star graph anomaly detection, 6) anomaly classification (core component), as well as 7) a mapping mechanism of anomaly data to the PenQuest meta model for additional threat semantics. Stages 3 and 4 provide anomaly scores that result in a binary malicious/benign classification, which can be used as additional feature in star graph anomaly classification (stage 6). Multi-class classification in preparation for model mapping is also performed in stage 6.

In this evaluation, anomaly classification is based on star structure anomaly reports. While it is also possible to use the computed anomaly traces of LLR and SEQUIN as training set, the respective components have proven to be better suited to process extraction and compression in the context of AIDIS. Nevertheless, LLR sentiment dictionary matching and SEQUIN's unsupervised anomaly detection capabilities have been evaluated as individual systems. Detailed results can be found in Chapter 5. In the following, we focus on the compound stages of the process and compare AIDIS to 3 similar solutions.

7.5.1 Experimental Setup

The prototype of the system was implemented in a test-bed environment consisting of 13 physical Windows 7 and Windows 10 computers used on and off by developers and IT personnel of a medium business over the course of half a year. The company, a local security solutions provider, performed regular checks to ensure that the machines in question were not affected by undesired software. One additional virtual Windows 7 instance was utilized for dynamically monitoring malicious software and automated targeted attacks on demand. All machines at least provided common user applications such as Microsoft Office, Adobe Reader, various browsers, as well as widely used OS extensions such as Java SE and the .NET framework. The required host and network event data was collected by a specifically developed kernel driver agent outlined in Section 7.4.1. Figure 7.5 provides an overview of AIDIS' testbed topology.

While the system is capable of collecting all the discussed event types, we omitted several of them (e.g. file reads) for practical reasons. Available event classes are listed in Table 7.5, represented as columns. The respective arguments considered were process, image, file, and registry path/key names, as well as accessed IP addresses denoted as hash values. The type of operation (table rows) was internally processed as numeric value ranging from 0.1 (registry read) to 3.5 (process termination).

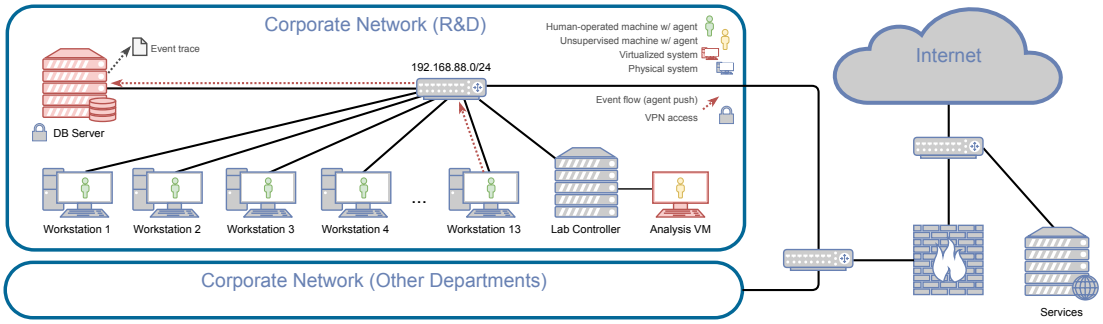


Figure 7.5: Topology of the testbed network. Event data is pushed to the database server in 3-second intervals, where it is converted to smart traces and made available for AIDIS processing.

	E	Process	Thread	Image	File	Registry	Network
Start	3.0	✓					
Stop	3.5	✓					
Spawn	0.9		✓				
Terminate	1.1		✓				
Load	1.5			✓			
Read	0.2				✗		
Create	0.75				✓		
Modify/Delete	1.25				✓		
Read	0.1					✗	
Set Key	0.5					✓	
Edit Value	0.25					✓	
Open Socket	2.0						✓

Table 7.5: Types of events collected by the agent and evaluated by AIDIS. The values for edge E were assigned manually for mapping purposes and in accordance to their approximated impact on the system. Operations marked with an ✗ are supported by the agent but were not considered in the evaluation.

The kernel monitoring agent logs all the event types to a central listener that in turn writes the events to a Postgres database server. SQL is used to query the database and to construct the star structures that are the basis for all further processing, which is handled by AIDIS' individual program components (see Section 7.5.2 for code specifics). Our approach is able to selectively retrieve entire system sessions or pick out individual processes, whereby any temporal range can be specified. For example, we can process only the first n seconds after an application's launch or extract data from a specific point within its lifetime – which is exactly what was done for our initial PoC evaluation ($n = 10$).

The repository of data included a total of 125 GiB of traces with more than 1.3 billion individual events across all monitored processes, with an event type distribution as depicted in Figure 7.6. Another 4.3 million (4.5 GiB) events were recorded on the aforementioned analysis VM. For these malicious traces, we executed a total of 1,995 APT malware samples and attack software, ranging from DarkComet [169] and other, unnamed Remote Access Trojans (RATs) to various crypto-miners and tools

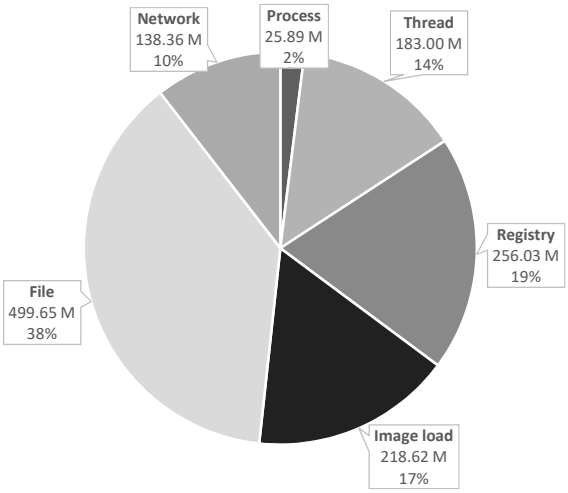


Figure 7.6: Types of events found in the full benign dataset. While 'process' events generally describe applications launched in the OS, the remainder represent actions triggered by said processes.

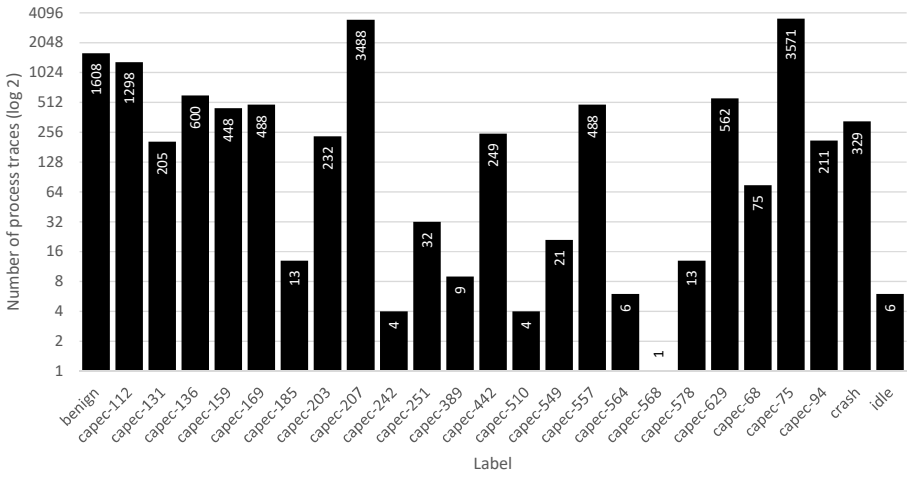


Figure 7.7: Number of process traces associated to a specific (CAPEC) class. The majority of data was labeled in accordance to its observed behavior, not malware family affiliation.

such as ShoulderSurfer[1], which is used for stealing information from Microsoft Exchange databases. Since AIDIS is not primarily used for malware classification but considers behavior independently, most monitored attack activity is not attributed to specific sample families. We instead use a CAPEC-based classification [219] to describe patterns for e.g. reconfiguring the system or disabling security mechanisms. This ensures that shared behavior is prioritized over family designation, which is typically not available for hitherto unknown samples. See Figure 7.7 for a distribution of classes used in the evaluation.

We performed only minimal cleanup of the input data by normalizing certain file paths and IDs (see Section 7.4.1). Largely idle or possibly faulty malware was retained,

[1]https://wikileaks.org/ciav7p1/cms/page_524353.html

since such samples are likely to be found in real-world datasets. The data was labeled from the get-go, assigning 22 CAPEC classes in addition to 3 meta categories:

1. BENIGN: Non-malicious execution of the process
2. CAPEC-112: Brute Force
3. CAPEC-131: Resource Leak Exposure
4. CAPEC-136: LDAP Injection
5. CAPEC-159: Redirect Access to Libraries
6. CAPEC-169: Footprinting
7. CAPEC-185: Malicious Software Download
8. CAPEC-203: Manipulate Registry Information
9. CAPEC-207: Removing Important Client Functionality
10. CAPEC-242: Code Injection
11. CAPEC-251: Local Code Inclusion
12. CAPEC-389: Content Spoofing Via Application API Manipulation
13. CAPEC-442: Malicious Logic Inserted Into Product Software
14. CAPEC-510: SaaS User Request Forgery
15. CAPEC-549: Local Execution of Code
16. CAPEC-557: Schedule Software To Run
17. CAPEC-564: Run Software at Logon
18. CAPEC-568: Capture Credentials via Keylogger
19. CAPEC-578: Disable Security Software
20. CAPEC-629: Unauthorized Use of Device Resources
21. CAPEC-68: Subvert Code-signing Facilities
22. CAPEC-75: Manipulating Writeable Configuration Files
23. CAPEC-94: Man in the Middle Attack
24. CRASH: Process crashed within 10 seconds
25. IDLE: Process shows insufficient activity for labeling

The numbering of the list corresponds to the class IDs seen in the confusion matrix below (see Table 7.8). Refer to the CAPEC repository [219] for more information about these particular attack patterns.

7.5.2 Code Implementation

The CSV-formatted graphs as well as the smart traces used by the sentiment component are preprocessed and converted into matrices using Bash and Python scripts. LLR-based sentiment analysis is implemented in R [251]. The optional grammar inference component of AIDIS is based on our own SEQUIN tool, which utilizes parts

of a Java Sequitur implementation by Eibe Frank[1]. The Hungarian distance (Kuhn-Munkres), which is the basis for all graph distance computations, is determined using the `solve_LSAP` function[2] available in R. Knowledge dissemination and the answering of competency questions is currently done via Linux on-board tools (see output in Section 7.4.5). For prototype-based template generation, we utilize a local Malheur [263] installation configured to accept non-MIST [314] input data. Similarity hashing is based on a Python tool coded by Chris McCormack[3]. Decision trees are computed in R using the `randomForest` function[4], while our SVM implementation utilizes `svm_Linear` and `svm_Linear_Grid`. The mapping of resulting anomaly reports and scores to the PenQuest model is currently done manually.

7.5.3 Results

Relevant Process Identification

Since this stage uses the sentiment analysis component detailed in Chapter 4, please refer to Section 4.5 for the evaluation of both relevant process extraction as well as the standalone LLR sentiment analysis system. In summary, we have identified the generic host process `svchost.exe` as the most viable candidate for ubiquitous kernel process monitoring.

SEQUIN Compression

Data compression is part of the 'grammar inference' component of AIDIS (see Figure 7.2). Like process identification, this stage is optional.

The standalone version of the SEQUIN component is evaluated in detail in Chapter 5. In summary, event trace compression resulted in a 97.2% reduction of data for the 51.3 million events that comprise the investigated (benign) `svchost.exe` traces. However, with 47.6% processing time reduction, the average speed-up of the star anomaly detection process was less than we have seen for smaller corpora, where the average speed increase for AIDIS-type data was around 73%.

Discussion – There are two factors that limit the use of SEQUIN as recommended stage in the AIDIS process: Firstly, memory consumption is significant for corpora above a certain size. Initial experimentation had us reach a 64 GiB ceiling at around 6 million events. While the more extensive experiment in the context of AIDIS consumed only 120 GiB RAM as opposed to the expected > 500 GiB determined by earlier linear regression (see Section 5.6.1), the memory demands on the analysis system remains a limiting factor.

[1] https://github.com/craignm/sequitur/tree/master/java
[2] https://www.rdocumentation.org/packages/clue/versions/0.3-55/topics/solve_LSAP
[3] https://github.com/chrisjmccormick/MinHash
[4] https://www.rdocumentation.org/packages/randomForest/versions/4.6-14/topics/randomForest

Secondly, the effective reduction of data may prevent the creation of star anomaly templates, as the amount of events remaining to compute meaningful prototypes is simply too small. In our specific case, both Malheur and MinHash failed to produce templates because of insufficient data.

This points to the conclusion that SEQUIN, while very effective in compressing star graph data, is better suited as visualization-assisted knowledge extraction system tasked with analyzing event data that cannot be accurately classified by AIDIS. By highlighting deviating events, SEQUIN helps to spot anomalies without relying on supervised learning. Because of the specific approach used by Sequitur [234], SEQUIN has proven to be best suited for processes that are less eclectic than e.g. `svchost.exe` in their default behavior, such as driver software, error handlers, and on-demand applications for specific user or system tasks.

In future research, we will also investigate SEQUIN's suitability as an additional feature in the anomaly classification process itself. Furthermore, the component's ability to extract common sequences warrants investigation into its suitability as alternative to the 'perfect match' approach to template generation. In the meantime, SEQUIN will be used to aid in analyzing outliers and non-ubiquitous processes.

Star Anomaly Detection

The anomaly detection process based on extracted star structures fulfills two major purposes: Firstly, it scores unknown process traces against one or several templates for a numeric anomaly score, and secondly, it provides a human readable report explaining the deviation from a baseline graph. These reports are then used in the anomaly classification component discussed in the next Section (7.5.3). In the following, we evaluate anomaly detection accuracy for the process `svchost.exe` using a single template produced by the 'majority' mode approach as well as a multi-template experiment utilizing 'prototype' mode followed by further reduction through similarity hashing (see Table 7.4 for an overview of graph template creation methods). Instead of analyzing traces in their entirety, we focus on the initial startup behavior, namely the first 10 seconds of execution of each process instance. In all cases, half of the available data was used for validation.

Results show that, while a single template is well suited for processes that exhibit stable behavior, it is difficult to accurately classify a versatile process like `svchost.exe` as benign or malicious with just one baseline to compare to. We ultimately achieved an accuracy of 89.34% using the mean benign score $\overline{m}$ as threshold separating the two classes. The optimum threshold was determined to be $\overline{m_o} = \frac{\overline{m}}{3}$. Using this value increased accuracy to 94.38%. Template generation took a negligible amount of time for each of the 13,961 malicious and 2,202 benign process instances, while matching required an average of 51 seconds per trace. See Table 7.6 for detailed results.

In a second experiment we automatically created a number of templates using the Malheur prototype approach, which resulted in 186 baseline traces computed in

Mode	Templates	Threshold	TP	TN	FP	FN	Accuracy
Single	1	$\overline{m}$	93.86%	37.04%	62.96%	6.14%	89.34%
Single	1	$\overline{m_o}$	99.99%	29.56%	70.44%	0.01%	94.38%
Multi	n	$\overline{m}$	93.86%	75.68%	24.32%	6.14%	92.42%
Multi	n	$\overline{m_o}$	99.99%	52.76%	47.24%	0.01%	96.23%

Table 7.6: Accuracy of the star anomaly detection component in standalone mode, for both single and multiple ($n = 17$) templates. We use 'majority' mode and 'prototype' plus 'similarity hashing' mode (reduction), respectively. While optimizing the threshold increases overall accuracy for the dataset, a more balanced approach to reducing the false positive rate is recommended (multi $\overline{m}$). Note that this AIDIS component is not typically used without subsequent classification, which boosts accuracy significantly.

around 30 hours. To reduce this number to more practical dimensions we used similarity hashing in 'reduction' mode, bringing this number down to 17 within a few seconds. Using similarity hashing without prior heuristic clustering did not prove feasible: With a processing time of 47 hours and a resulting 589 templates, it was less computationally effective and yielded too high a number of templates for practical use.

To evaluate the multi-template approach, we considered the same dataset as before. If any of the 17 benign templates deemed a trace as within tolerance, the process run was classified as non-malicious. This increased the overall accuracy to 92.42% when using the arithmetic mean of benign scores $\overline{m}$ as threshold, while $\overline{m_o}$ accuracy was boosted to 96.23%. The false positive rate was significantly reduced from 63% (70.4%) to 24.3% or 47.2%, respectively. For detailed results, see Table 7.6.

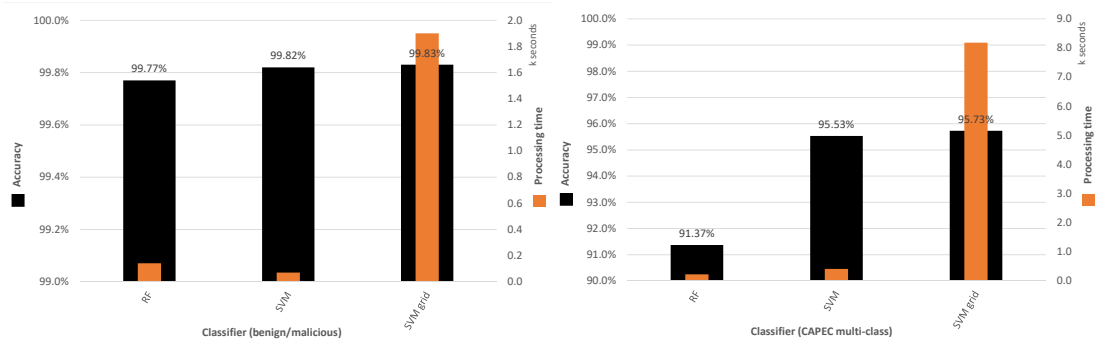
Discussion – Above results show the 'worst-case' accuracy of the star anomaly detection process when used as an individual system. Despite the better overall numbers, we recommend the mean or median benign score from the training set as threshold between the benign and malicious classes, as it more drastically reduces the false positive rate.

The key outcome of this evaluation stage isn't stand-alone component accuracy, but the degree of *process coverage*: 88.48% of all malicious activity created events attributed to `svchost.exe`. Considering single-template matching using $\overline{m}$, this resulted in a 83.05% accuracy in attack detection when observing only the generic host process for a total of 10 seconds. Combined with subsequent anomaly classification (see Section 7.5.3 below), this number was pushed to 88.31%. This finding is especially promising as it might help eliminate the need to incipiently identify malware binaries and to observe more that a few key OS processes such as the ones identified in Table 4.3) for system-wide attack detection.

In conclusion, it stands to mention that threshold-based anomaly detection it not AIDIS' core purpose. While the results are workable, the overall false positive rate is still too high to trust this purely number-based decision. As initially argued, considering event semantics is key to improving detection rates and eventual interpretation, which is why the actual classification of anomalies identified in this stage is considered the system's main component, discussed hereafter.

Classifier	Classes	OOB error	Accuracy	Kappa	C-value	Time (s)
RF	2	0.26%	99.77%	-	-	142.1
RF	n	4.96%	91.37%	-	-	224.8
SVM	2	-	99.82%	99.24%	1	70.4
SVM grid	2	-	99.83%	99.28%	0.25	1899.8
SVM	n	-	95.53%	94.67%	1	412.0
SVM grid	n	-	95.73%	94.87%	1.75	8180.4

Table 7.7: Classification accuracy ($n = 25$) of the RF and SVM approach. Support vector machines generally proved to be more accurate in our scenario. For Random Forest, we tried 1000 trees with 100 variables on each split. For SVM, we used 10-fold cross validation with 3 repeats to reduce overfitting.



(a) Binary classification accuracy

(b) CAPEC classification accuracy

Figure 7.8: Classification accuracy and processing times overview for binary and multi-class RF/SVM. The use of hyperplane optimization through the alteration of the C-value slightly improved the results, but significantly increased processing times, making common linear kernel SVMs the most sensible choice for our dataset.

Star Anomaly Classification

With the previous star anomaly detection stage providing a purely threshold-based score, anomaly classification takes the resulting reports (as shown in Section 7.4.5) and asks a number of competency questions, the answers of which are used as classification features. We tested two technical approaches to classification: Random forest, and linear support vector machines with and without variable C-score (hyperplane optimization).

In the first experiment, we used previously generated single-template anomaly reports from the anomaly detection stage and classified the data into a benign and malicious category: Both methods returned near-perfect results, with a ROC accuracy of 99.77% for RF and 99.82% accuracy for SVM (see Table 7.7 and Figure 7.8a for details).

Finally, we repeated the process with the labeled data of the aforementioned 13,961 malicious and 2,202 benign process instances in order to test classification into 22+3 CAPEC-determined behavior categories. Linear kernel SVM with a C-value of 1.75 achieved the best result with an accuracy of 95.73%, closely followed by a constant-C ($C = 1$) SVM with 95.53% and Random Forest with 91.37% multi-ROC accuracy and an out-of-bag (OOB) error rate of 4.96%. See Table 7.7 and Figure 7.8b for a detailed

overview of classification accuracy as well as processing times. Table 7.8 shows the class confusion matrix of the most accurate approach.

Discussion – Above results show the advantage of semantics-enabled classification over purely threshold-based approaches. Even with 25 classes, the accuracy was much higher than with the default ($s_t = \overline{m}$) benign/malicious distinction used in the previous stage. There is still room for improvement, however. The main source of misclassification were CAPEC patterns 75 [1] ('Manipulating Writeable Configuration Files') and 207 [2] ('Removing Important Client Functionality'). Here, between 4 and 8% of the associated traces were misclassified as the respective other. The reason can be found in the ambiguous nature of the pattern as well as the difficulty of clearly labeling data as one or the other: The removal of client functionality (e.g. firewall or user authentication measures) is often done by manipulating configuration files, leading to similar star graph elements and by extension, anomalous events.

Key features as per mean decrease in accuracy/Gini turned out to be the count of error function libraries imported, the use of Windows user management and universal app functions, high (system) registry interaction, operations related to log files, network activity in general, as well as the import of data access and diagnostics functions. See Figure 7.9 for a list and explanation of the most impactful features.

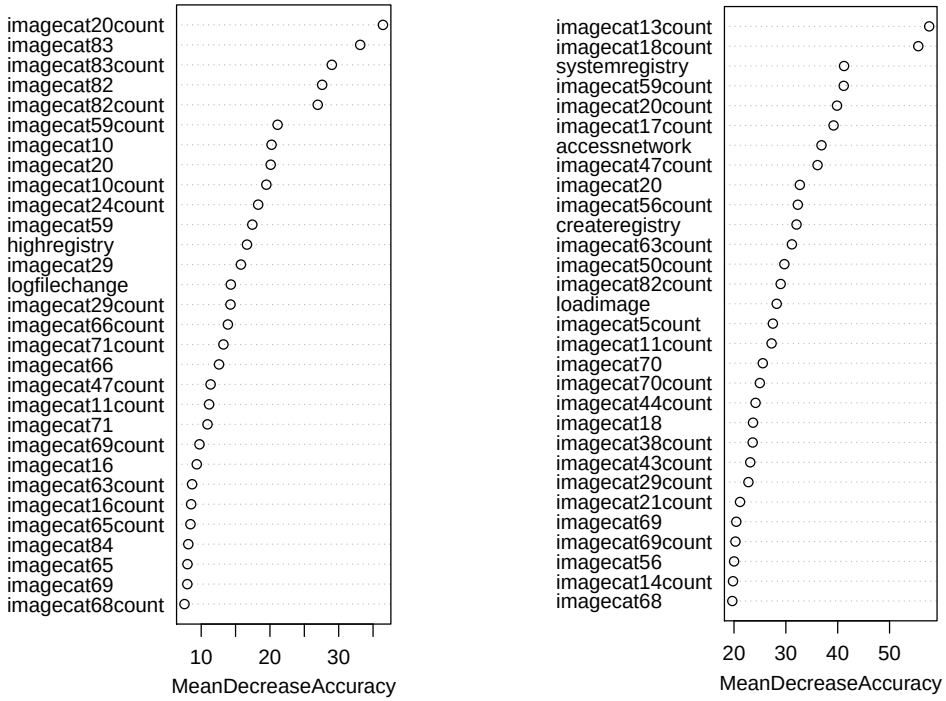
Summed up, many of the most relevant features are related to image load and registry operations. The reason for this can be found to a degree in the selection of data used in the experiment: With a focus on the initial 10 seconds of activity, it is expected to see numerous events pertaining to the dynamic linking of libraries [105], which is generally more widely used than static or runtime linking in both malware and benign software. The concept of dynamic linking requires applications to search and load library (DLL) resources at launch, resulting in a spike of corresponding 'image load' events. Registry events typically represent initialization tasks or changes to certain settings, something that is often seen in the early stages of operation as well. Interestingly, file events were found to be generally underrepresented during the start-up of compromised process instances in particular: Only 108 events in the selection described malicious file operations, as opposed to 40,538 events in the benign `svchost.exe` corpus. As a result, the lack of specific file operations might be a strong indicator of manipulation – something that has to be considered in future feature selection. Currently, only the general existence of such events is assessed.

While the importance of image and registry operations constitutes an interesting finding in itself, upcoming experiments will increasingly focus on file and network events typically triggered during a process's entire lifetime. This includes defining new features corresponding to questions about the nature of any created or modified files, as well as to the fact that compromised system processes fail to display a level of file interaction commonly seen in their benign cousins.

[1] https://capec.mitre.org/data/definitions/75.html
[2] https://capec.mitre.org/data/definitions/207.html

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**
1	1148	0	0	0	0	0	0	0	0	0	0	0	0	4	1	0	0	0	0	0	0	0	0	0	0
2	0	380	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	70	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	1	0	0	150	0	0	0	0	3	0	0	0	0	0	0	9	0	0	0	0	0	8	0	0	0
5	0	0	0	0	136	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
6	0	0	0	1	1	130	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	2	0	0	0
7	0	0	0	0	0	0	2	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0
8	0	0	0	0	0	0	0	62	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
9	1	0	0	1	0	1	0	3	1032	0	0	1	0	0	0	3	0	0	0	0	0	78	0	0	0
10	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	3	1	0	0	0	0	0	0	0	0	0	0	1	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	81	0	0	0	1	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	2	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	128	0	0	0	0	0	2	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	3	0	0	0	0	1	0
20	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	154	0	2	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	24	0	0	0	0
22	0	0	0	1	0	2	1	1	38	1	1	0	0	0	0	1	0	1	0	5	0	977	0	0	1
23	0	0	0	2	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	2	60	0	0
24	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	3	1	94	3
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 7.8: Confusion matrix for SVM with linear kernel, C-value of 1.75. The validation resulted in a 95.73% accuracy and a Kappa statistic (comparing observed to expected accuracy) of 94.87%. Classes with the highest misclassification rate were 9 (CAPEC-75) and 22 (CAPEC-207).



(a) Benign/malicious classification

(b) CAPEC category classification

Figure 7.9: Overview of features linked to the answer of a corresponding competency question. Features prefixed by 'imagecat' correspond to observed image load ('loadimage') activity (or the count thereof) in one of the 87 Windows library (DLL) categories parsed from various Microsoft and developer sources as part of the initial project stages. 'createregistry' and 'highregistry' determine the existence of anomalous operations that insert data into the Windows registry and the presence of a large number (> 35) of create/change/delete operations in general. 'systemregistry' is one of the few fixed pattern questions that are set to 'true' when keys within the `HKLM\System` registry hive are interacted with. 'logfilechange' does the same when `*.log` or `*.evt(x)` files are modified. If the parsed anomaly report contains network interaction, the 'accessnetwork' feature value is set to 'true'.
Image categories in the top 10 features: COM (10), Data Access (13), DHCP/DNS (17), Diagnostics (18), Error Handling (20), File System (24), Remote Desktop (56), Security (59), Windows User Management (82), Windows Universal App (83). A full list of categories and libraries within is available on request.

Model Mapping

Armed with the automated classification of anomalies as belonging to a specific CAPEC pattern, it now becomes possible to link our data with the PenQuest meta model for further semantic enrichment, interpretation, and mitigation planning.

For our evaluation, we use the class with the most events while boasting a low misclassification rate. Disqualifying classes 9 and 22 (see Table 7.8), we take a look at class 2 (CAPEC-112 [1], 'Brute Force'), with a total number of 380 process anomaly reports and a misclassification rate of 0%.

For data mapping, we use PenQuest's $\langle Event\langle Type = Anomaly\rangle\rangle$ notation of X, as specified in Sections 7.3.3 and 6.5. With a time range of $\langle Time\langle Start = 0, End = 10\rangle\rangle$, each $\langle Operation\rangle$+$\langle Argument\rangle$ pair with $\langle Parent = svchost.exe\rangle$ will be appended in sequence, resulting in a simple description of X that can be easily converted to other threat definition languages or shared directly with others.

The link to CAPEC provides us with additional semantic information, namely that 'Brute Force' refers to activity where the "attacker attempts to gain access to this asset by using trial-and-error to exhaustively explore all the possible secret values in the hope of finding the secret (or a value that is functionally equivalent) that will unlock the asset." [219]

According to the meta model and Figure 6.7, CAPEC-112 can be an APT kill chain 'Delivery–Intrusion' as well as an 'Installation–Propagation' or 'Installation–Persistence' support action which is typically used in combination with other attack activity. Categorized as the identically named 'BF' (Brute Force) attack action, PenQuest also defines appropriate primary controls countering the threat, namely 'Authentication protection' (AP): This controls group is primarily concerned with managing authenticators such as passwords, tokens, and biometric information. Associated defense actions include the categories 'Remote Access' (REA) and 'Authenticator Management' (AUM), with a range of controls directly out of NIST SP 800-53 [146]. Specific countermeasures therefore include remote access control and encryption, access point management, password/PKI/hardware/biometric authentication, as well as controls related to cache expiration settings.

The information gleaned from the model can now be used to plan appropriate defensive measures to prevent this particular attack. PenQuest's gamified nature also allows us to play through the attack and test various controls and systems that may reduce threat impact and probability. While the efficacy of the suggested countermeasures need to be evaluated on a case-by-case basis using real world infrastructure, 9 interviewed security practitioners of intermediate, professional, and expert level strongly (3 points) or rather agree (2 points) that the model is 'applicable to real world scenarios', resulting in 21 out of 27 possible points. Almost all testers strongly agree that

[1] https://capec.mitre.org/data/definitions/112.html

using PenQuest increases general security awareness when used (22/27 points). Refer to Chapter 6 for more information on the model's in-depth evaluation.

Discussion – It stands to reason that this final mapping stage of AIDIS is difficult to quantitatively evaluate. Future work will investigate crafted attack scenarios that are then mitigated by specific defense measures suggested by the respective NIST categories in order to determine the effectiveness of the controls. Furthermore, we will design a full simulation component of the gamified model using reinforcement learning to automate the process of 'playing through' a large number of attacks for strategy optimization purposes.

On the modeling side, not all of the 517 CAPEC classes and only a portion of the defensive controls are currently part of PenQuest. Around 12% of the available patterns have been used to populate the model to date, whereas 'detailed' technical patterns referring to specific software attacks (as opposed to 'meta' and 'standard') are not currently included. Control-wise, we prototypically implemented 70 out of 224 controls specified by NIST. With the model itself ready for use, the remainder of the data can be added at will. However, it needs to be stated that the CAPEC repository itself is missing some of the required information needed for successful automated mapping, which increases the effort required to add the remaining patterns. For future iterations, we will therefore consider alternative vocabularies such as MITRE ATT&CK[1].

7.5.4 Comparison

In this final evaluation section we take a look at 3 systems that share technical aspects with AIDIS' core components. Please note that it is generally difficult to compare our approach to any alternative solutions, since the data basis used for training and validation is not the same for reasons of both availability and compatibility. Furthermore, none of the identified works provide semantic enrichment through a model such as PenQuest.

In the following, we pit AIDIS against a general intrusion detection system based on similar SVM multi-class classification [10], as well as against two graph-based threat detection systems [12, 141] using binary classification. Refer to the 'Related Work' section for additional information about the discussed works. Figure 7.10 provides an overview of the respective accuracy scores. Additional in-depth comparisons of the overall system can be found in Section 8.2.1.

SVM Multi-Class Classification for Network Traffic

With its SVM-based multi-class classification based on the KDD99 dataset [318], Ambwani [10] presents a network-based intrusion detection system attempting to distinguish 4 different attack categories: denial of service (DoS), probing, remote to local (R2L),

[1] https://attack.mitre.org/

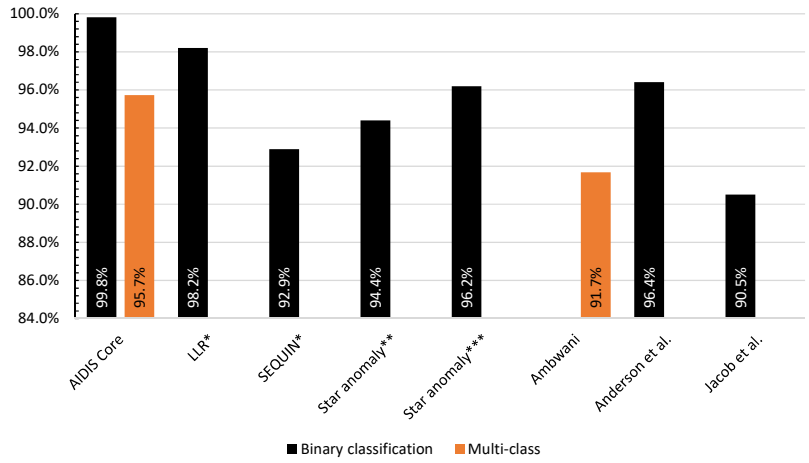


Figure 7.10: Classification accuracy of AIDIS components (both core and stand-alone) compared to three similar systems [10, 12, 141].

and user-to-root (U2R). With a total of 23 inherent attacks considered, the classification problem is comparable to AIDIS in scope. Its focus on full traffic dumps makes Ambwani [10]'s solution a direct network equivalent to our host-based approach, which uses both SVM and RF in its classification of predetermined anomalies.

The authors observed an optimized 91.67% detection accuracy when discriminating 23 classes. AIDIS achieves 95.73%, which marks a significant improvement over a purely network-traffic based system. While this does not make Ambwani [10]'s approach obsolete in any way, it gives a strong indication that assessing endpoint events is at least as feasible as using traffic dumps when it comes to ML-powered intrusion detection. Coupled with AIDIS' anomaly detection stages, our system additionally provides means to spot unknown behavior not covered by labeled datasets.

Graph-based Attack Detection

Since AIDIS utilizes a graph-based approach, we compare it to two such systems in our evaluation. Previously outlined in Section 7.2, the work by Anderson et al. [12] encompasses a detection algorithm based on the analysis of graphs constructed from dynamically collected instruction traces. The utilized SVM classifier is tested in a binary scenario distinguishing benign from malicious software. With a combined kernel (best-case) accuracy of 96.41% compared to AIDIS' 99.82%, the solution fares notably worse. In terms of performance, however, our system is at a disadvantage, as it requires an additional ∼50 seconds per trace instance to compute the anomaly graphs needed for classification.

Jackstraws [141] offers host-side C2 traffic identification through dynamic analysis of individual malware samples. Not unlike AIDIS, it models data flows between API calls as graph. The resulting patterns are used as templates for subsequent detection. This is also the key difference to our solution: Under the hood, Jackstraws extracts graph templates not as baseline for anomaly detection, but uses them as de-facto signatures.

This results in a high number (75%) of unclassified network connections. The remainder was categorized with workable accuracy, but still scores over 9 percentage points lower than AIDIS core in a binary classification setting.

7.6 Discussion

In this section, we want to highlight current limitations of the AIDIS system and discuss three main areas of improvement: Automation, performance, and accuracy.

Regarding *automation*, our approach currently relies on the manual definition of competency questions that provide the features for RF/SVM classification. Especially thresholds (e.g. what constitutes a 'high number' of file or registry events) stem from software analysis experience rather than statistical evaluation. Here, future work will improve and automate both the creation of the competency questions as well as the process of parsing anomaly reports, which is currently realized through conventional text processing scripts. Another area that would benefit from additional automation is the conversion and mapping of classified anomalies to the PenQuest notation. This will help to ultimately map monitoring data to an ontology describing the meta model, which is currently a work in progress based on earlier efforts such as TAON [188]. Future research will also include the automated simulation of attack scenarios aimed at discovering optimal defense strategies through reinforcement learning.

A second aspect in need of improvement is *performance*. Currently, the creation of graph templates and their matching to target star structures is implemented as an early prototype that does not significantly optimize these computationally complex operations. Future iterations of AIDIS will therefore focus particularly on runtime reduction to open the door for close to real-time applications. This will also include optional stages such as sentiment analysis and Sequitur compression, where the resolving of rules is responsible for much of its processing overhead (see Chapter 5).

Lastly, there is the matter of *accuracy*. While the general results are promising, star structure anomaly detection by itself is in need of further fine-tuning to bring down false positive rates for multifaceted processes such as the investigated Windows host process. However, acknowledging that threshold-based systems are unlikely to achieve the same level of accuracy as semantic approaches, most effort will be invested into improving star structure classification for multiple categories. Firstly, we will investigate potentially less ambiguous and more complete vocabularies than CAPEC to reduce the risk of misclassification. Secondly, we will develop new competency questions utilizing the insight gained from evaluating AIDIS – especially the determined decrease of accuracy for various question types (Figure 7.9) and the observed peculiarities of (malicious) file events. This will help replace some of the less relevant questions with more expressive ones. Future versions of AIDIS will also see the inclusion of other detection system scores/results as additional classification features, which is expected to further improve accuracy.

For a more in-depth look at limitations, refer to Section 8.2.2 in the concluding chapter.

7.7 Summary

In this chapter, we presented the core components of AIDIS, a star structure-based "Advanced Intrusion Detection and Interpretation System' able to detect and explain anomalous deviations in operating system process behavior. The returned output of detailed state changes as well as a tendency towards a specific APT stage and attack pattern is expressed through the mapping of semantic key factors to PenQuest, our dedicated attacker–defender model (Chapter 6). At the same time, the model suggests specific measures intended to counter any observed attack. Specifically, this chapter contributed by:

- Presenting a holistic approach to collecting and analyzing host and network events able to describe and assess all victim-side APT attack stages;
- Introducing a transparent anomaly detection system based on star structures;
- Providing optional processing components incorporating features such as event sequence compression through grammar inference and sentiment analysis for identifying expressive operating system processes;
- Classifying anomalies into semantic threat categories based on the CAPEC dictionary [219] using both a Random Forest (RF) and Support Vector Machine (SVM) approach;
- Enabling the interpretation of classified data through a comprehensive targeted attack meta model encompassing actors, assets, as well as hostile and mitigating actions.

The process was prototypically implemented and successfully tested using real-world process data captured on more than a dozen company workstations over half a year. Ultimately, 99.8% of all star structure anomalies were correctly identified as benign or malicious, with a solid 95.7% accuracy in multi-class scenarios that seek to associate each anomaly with a distinct CAPEC attack pattern. Furthermore, we have shown that 88.3% of close to 2,000 attacks could be accurately identified by observing and classifying just one generic Windows process (`svchost.exe`) for a mere 10 seconds, thereby eliminating the necessity to monitor each and every (unknown) process existing on a system.

In comparison to similar solutions, our system promises improved accuracy for both binary and multi-class classification, as evidenced in Section 7.5.4. When pitted against research identified in the literature survey, AIDIS sets itself apart through high accuracy and its core features in general – namely its classification, modeling, and white-box anomaly detection capabilities not reliant on fixed patterns. See Section 8.2.1 for more details.

AIDIS represents the core contribution of this thesis by providing transparent anomaly detection and classification for ubiquitous kernel processes. For future research (see Section 8.2.3), we aim to focus on strategy inference utilizing PenQuest's model in combination with anomalous events tagged by AIDIS's other components to compute optimal responses to a wide range of attacks. This will enable analysts to employ our solution as expert system supporting both risk management and organizational threat mitigation while being provided detailed technical assessments about individual stages of an intrusion.

Chapter 8

Conclusion

Contents

This chapter summarizes the evaluation of all technical AIDIS components and discusses individual and combined accuracy. Our system is additionally compared to 13 key works identified during literate review, highlighting differences, commonalities, and research opportunities. A closer look is taken at processing times and their implications for productive deployment. Research questions, hypotheses, and overall contributions are revisited and we outline future work aiming to improve data procedures, event processing, computational performance, and overall accuracy. Last but not least, we discuss optimal strategy inference through model checking and reinforcement learning as a promising direction of further research.

8.1 Overall Evaluation

The Advanced Intrusion Detection and Interpretation System is very flexible in terms of implementation: The individual components can be deployed alone or in concert – promising a wide range of possible applications within an organization. This section summarizes the evaluation outcomes previously discussed in the respective evaluation chapters and highlights where and how each AIDIS component is best utilized. Finally, we centrally answer the research questions and test the hypotheses stated in Section 1.3.

8.1.1 Results Summary

Accuracy

Next to the compound system dubbed 'AIDIS Core' (Chapter 7), all individual components can be employed in standalone mode and have been tested with standard event traces as described in Section 3.3. With its focus on compression and more exotic anomaly extraction scenarios, the SEQUIN prototype is the only exception to this rule: Here, ICS sensor data was analyzed in addition to individual OS traces.

Each system's performance was assessed using the traditional accuracy score as main performance indicator. According to ISO 5725-1 [101], the term "accuracy" is used to describe the closeness of a measurement to the true value. It is defined as:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

$$\begin{aligned} \text{where } TP &= \text{true positive rate,} \\ TN &= \text{true negative rate,} \\ FP &= \text{false positive rate,} \\ FN &= \text{false negative rate.} \end{aligned}$$

The reason for this choice over alternatives such as the f-measure [249] is its widespread use in comparable solutions and the fact that many machine learning applications utilize traditional accuracy by default.

Figure 8.1 shows the accuracy values determined for each component of AIDIS. We specifically evaluate the LLR sentiment analysis system, 4 different template and threshold optimization variants available in the star structure anomaly detection system, SEQUIN's grammar inference component, the PenQuest model, and the combined system (AIDIS Core) as a whole.

For both binary and CAPEC multi-class classification, *AIDIS Core* utilizing SVM with hyperplane optimization offers the highest accuracy with 99.82% and 95.73%, respectively. Considering processing times, the more viable choice in both cases is conventional linear kernel SVM, however: Here, scores are only slightly lower but computation time decreases by at least the factor of 20. Figure 7.8 provides further details.

As a standalone system, the *LLR* component performed well in comparison to similar n-gram-based implementations such as AccessMiner [176], which achieved an average accuracy rate of 89.5% using file operation trigrams of benign applications as baseline. For registry activity, the results of AccessMiner are far inferior (48.6%). This supports our decision of observing several event types in synergy and of basing our decision on a statistical test rather than the number of n-grams deviating from a baseline. In our experiments, LLR even yielded better results than each of the four star structure algo-

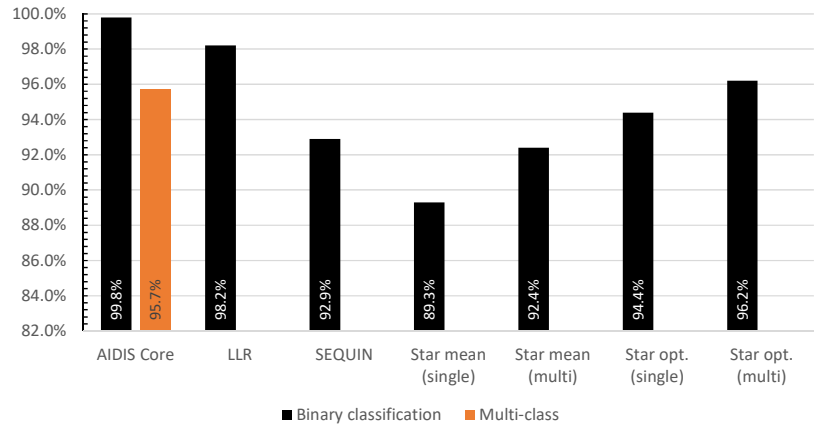


Figure 8.1: Classification accuracy comparison of the AIDIS core system and its individual components. Only the star anomaly classification system supports the assignment of multiple classes.

rithms. The reason for this can be found in the different approaches to data processing: LLR assesses bigrams of events that maintain their strict chronological order. While this makes each pair more expressive than the individual events processed by the star structure system, it also makes the algorithm more susceptible to mimicry attacks that inject events into a process or thread, thereby interrupting sequences that would normally constitute a malicious bigram. For star structures, the order of events does not strictly matter in the context of a specific process – which might risk misclassification of ambiguous activity but increases resilience to obfuscation. Ultimately, the choice of system depends on the data analyzed: The more volatile a process, the more our preference shifts towards the use of star structures, where we can opt to use multiple templates instead of a single dictionary.

Star anomaly detection by itself resulted in a workable accuracy of at least 89.3% and up to 96.2%, depending on the number of templates and the kind of threshold optimization used. Utilizing multiple templates for complex processes such as `svchost.exe` is highly recommended: In our experiments, doing so increased accuracy by at least 2 percentage points across the board. Still, optimizing the threshold instead of using the mean star graph distance deviation should be used sparingly to avoid overfitting. The key contribution of the (standalone) star anomaly system is undoubtedly the new insights into process coverage: Our results show that 88.48% of all malicious activity is reflected in the first 10 seconds of activity of the `svchost.exe` generic host process. This finding has the potential to significantly contribute to IDS efficiency for both new and existing solutions. Lastly, the star structure component provides the anomaly information needed for the classification happening in AIDIS Core – its promising results would not be replicable without this semantic data.

The final technical component, *SEQUIN*, fulfills a special role. Next to data compression, the Sequitur-based system is the only one that offers unsupervised learning for anomaly detection. Frequent and anomalous patterns can be extracted and explored in detail. When tested at scale, SEQUIN achieved a detection accuracy of 92.9% on ICS

Process	Edge	Event
cmd.exe	106	rule-329
httpd.exe	105	rule-67
cmd.exe	3	process-conhost.exe
SYSTEM	2	network-private-ip
httpd.exe	105	rule-67
cmd.exe	106	rule-510
httpd.exe	116	rule-277
httpd.exe	3	process-cmd.exe
cmd.exe	3	process-conhost.exe
httpd.exe	106	rule-21
httpd.exe	0.75	file-\dev\harddiskvolume2\xampp\tmp\php6c6.tmp
httpd.exe	1.25	file-\dev\harddiskvolume2\xampp\tmp\php6c6.tmp
httpd.exe	0.75	file-\dev\harddiskvolume2\xampp\htdocs\phpfilemanager-0.9.8\jurex\juril.php
httpd.exe	1.25	file-\dev\harddiskvolume2\xampp\htdocs\phpfilemanager-0.9.8\jurex\juril.php
httpd.exe	1.33	file-\dev\harddiskvolume2\xampp\tmp\php6c6.tmp
httpd.exe	107	rule-66
cmd.exe	3	process-conhost.exe
httpd.exe	105	rule-67
cmd.exe	3	process-conhost.exe
httpd.exe	112	rule-76
httpd.exe	0.75	file-\dev\harddiskvolume2\xampp\tmp\sess_0ipkv6fuv51pa6eqj9slfo0cr5
httpd.exe	0.75	file-\dev\harddiskvolume2\xampp\tmp\sess_r8u0graeflq017fbo5mlad56s1

Table 8.1: Example SEQUIN output highlighting anomalous entries in a sequence of Windows events. The data follows the format of $G = (U, V, E)$, as explained in Section 7.4.5. Dark grey lines represent rules (recurring events) identified by SEQUIN. Light grey marks temporary files, black represents anonymized network activity and orange highlights uninterrupted blocks of 4 or more anomalous terminals.

sensor data, with a 100% detection rate for undesired behavior in general. Additional experiments with Windows event data successfully identified a reverse shell web server attack through SEQUIN's statistical data evaluation coupled with visual analytics. Table 8.1 depicts an extract of the result.

The *PenQuest* model was evaluated in three ways: First, the game prototype was play-tested with 9 security practitioners of intermediate to expert knowledge level in order to assess its suitability for threat modeling, risk assessment, education, and awareness training. All participants agreed that the experience of using PenQuest was overly positive. Its applicability to awareness training was met with uniform agreement. Similarly, most testers strongly or rather agreed that the model is 'applicable to real world scenarios' and that PenQuest increases security insight when consulted. Detailed information about test and questionnaire can be found in Section 6.6.2.

The second part of the evaluation determined if the model manages to provide a strategy set for all APT stages as well as all three aspects of the CIA triad, while providing a balanced and realistic range of attack patterns and possible countermeasures. Results show that the gamified model is mostly complete in its coverage: Only 'state attacks' such as race conditions and certain kinds of session forging remain underrepresented because of labeling deficiencies in the CAPEC repository – something that can be easily remedied by manually adding new actions. At the same time, our populated model generated or confirmed insights into the current state of organizational defense

	ID	Processing step	Processing time (s)
C	1	Preprocessing	51,753
LLR	1	LLR calculation	235,281
	2	LLR dictionary computation	22,684
	3	LLR key process extraction	1,932
S	1	SEQUIN inference (standalone)	51,212
Star template	1	Stars extraction (all processes)	71,971
	2	Stars template generation (match)	123,070
	3	Stars template generation (majority)	123,509
	4	Stars template generation (prototype)	109,465
	5	Stars template generation (similarity)	40,143
	6	Stars template generation (prototype+similarity)	6
	7	Stars template generation (Sequitur+match)	7,328
	8	Stars template generation (Sequitur+majority)	7,332
	9	Stars template generation (Sequitur+prototype)	n/a
	10	Stars template generation (Sequitur+similarity)	452
Star anomaly	1	Stars anomaly detection	833,389
	2	Stars anomaly classification (question parsing)	20,214
	3	Stars anomaly classification (RF binary)	142
	4	Stars anomaly classification (RF multi-class)	224
	5	Stars anomaly classification (SVM binary)	70
	6	Stars anomaly classification (SVM binary w/ hyperplane opt.)	1,900
	7	Stars anomaly classification (SVM multi-class)	412
	8	Stars anomaly classification (SVM multi-class w/ hyperplane opt.)	8,180

Table 8.2: Overview of processing times including all alternative template generation and classification methods for process `svchost.exe`. For compound steps (denoted as $A + B$), only phase B duration is listed. Sequitur compression followed by 'prototype' template generation could not be measured since there was insufficient data for Malheur prototype extraction.

even without mapping data to a concrete scenario: For example, awareness initiatives consistently counter a high number of attacks across multiple categories. Similarly, media storage guidelines, security engineering principles, and cryptographic controls tend to have greater reach and promise organization-wide benefits. For more information about strategy set distribution and insights, see Section 6.6.2.

Lastly, we investigated the mapping of individual attacks identified and classified by AIDIS Core. This final random sample test complemented the evaluation of the game component and semantically assessed if the countermeasures suggested by PenQuest 'make sense'. An example of such a test can be found in Section 7.5.3. Overall, decision support functioned as intended and yielded sensible suggestions even though the model is only partially populated with data. Additional large-scale testing with security operators needs to be conducted in the future to determine whether data sources other than CAPEC and NIST SP800-53 synergize well with PenQuest. The gamified ruleset itself is fully capable of incorporating alternative vocabularies with negligible effort.

In the following we take a closer look at the second major appraisal factor: Computational performance.

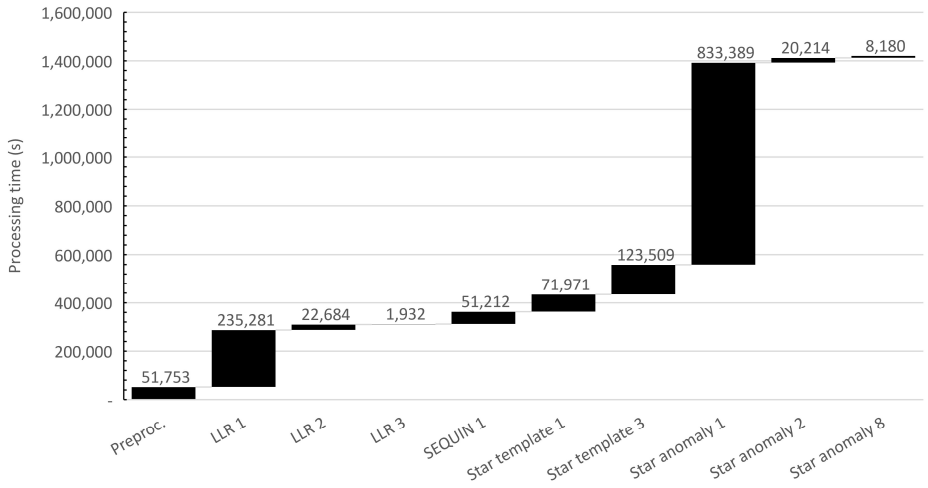


Figure 8.2: Processing times of the main evaluation for linear kernel SVM (grid) multi-classification of process `svchost.exe`. Note that steps 1 through 6 do not need to be repeated regularly and that expensive step 7 is rarely conducted for more than a few new process instances at once. The chart here depicts computation times for *all* available data (16,361 `svchost.exe` traces) collected over 6 months.

Performance

While the AIDIS prototype is far from optimized, the overall processing times are still indicative for its future use in productive environments. Table 8.2 lists all mandatory and optional steps for the overall performance evaluation. LLR stages (1) through (3) determine the most relevant processes for observation and therefore need to consider all available traces. With the exception of the stars extraction step (1), the remainder of processing times are limited to `svchost.exe`.

We can see that the most expensive stages with a computation time of above 100k seconds are the LLR calculation, the stars template generation, and the stars anomaly detection process. Thanks to SEQUIN, which offers efficient compression of most input data, it is possible to reduce corpus sizes by up to 97.2%. This cuts down star structure processing times by 47.6% to 73%. LLR calculation times can be further reduced by approximately 90% if future runs are limited to identified processes of relevance in the first place. Moreover, stars template generation is a step that only needs to be repeated on demand upon an expected change to an application's baseline behavior (e.g. because of new software rollouts). Additional re-training may be necessary at regular intervals of weeks to months to counter concept drift [315].

Since many processing steps listed in Table 8.2 are optional or represent alternatives for generating templates, Figure 8.2 depicts the computation times specific to our main evaluation scenario. Star anomaly detection is undoubtedly the most impactful factor: With a mean computation time of 50.9 seconds per `svchost.exe` process trace, anomaly detection is not quite realtime-capable in AIDIS' current iteration. Other processes compute significantly faster but are not as expressive because of their limited function scope within the OS. For example, the process for changing file attributes (`attrib.exe`), the

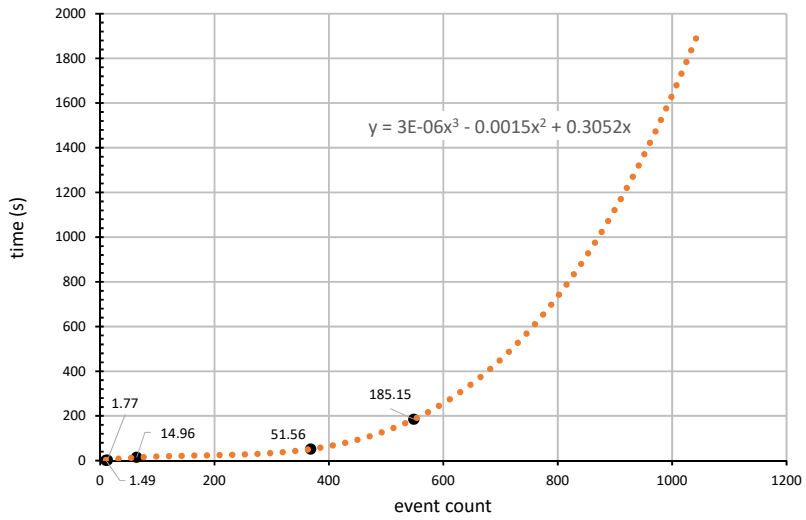


Figure 8.3: Regression analysis of different-length event traces. With the star graph's polynomial scaling in time, traces with more than 700 events will take approx. 10 minutes to complete. The current `svchost.exe` average for a time span of 10 seconds is 368 events, processed in 51 seconds.

configuration tool `net.exe`, and the session manager `smss.exe` all completed anomaly detection in less than 2.5 seconds per trace. Observing `svchost.exe`, an organization deploying AIDIS would have to invest approximately 1.69 hours of mid-range CPU time (2.4 GHz, single-core processing) each day to calculate anomaly scores for a network of 10 computers, provided the machines are used by a single user adhering to average FTE work hours.

While future code optimization is expected to reduce these numbers significantly, it is important to keep in mind that increasing the observed time span to more than the currently analyzed 10 seconds of process activity will further increase processing times at a polynomial rate (see Figure 8.3). It is therefore unlikely that a process as complex as the Windows generic host can be timely assessed for more than a few minutes of runtime without using a sliding or session window approach. Please refer to Section 8.2.2 below for a discussion about how this alternative take on data processing might look like.

AIDIS' accuracy even before optimization makes it a promising alternative to existing systems, which rarely offer the same level of functionality. In the following, we take a look at the system's key contributions and compare it to solutions identified as useful additions to an organization's APT detection efforts.

8.1.2 Contributions

In the following, we revisit the problems identified in Section 1.2 and summarize how our proposed solutions contribute to the state-of-the-art in threat detection.

There are five aspects we have identified as key contributions of AIDIS: The non-reliance on fixed patterns or signatures, independence from pre-extracted (known) sam-

ples, a white-box approach to anomaly detection, support for anomaly classification, as well as the existence of an underlying threat model that enables wide-ranging interpretation capabilities. Specifically:

Pattern-free anomaly detection. All components of AIDIS have been designed with pattern avoidance in mind. There are no signature-based mechanisms in place. For anomaly detection or classification, none of the components utilize heuristics or other behavioral patterns as part of their detection routines:

- LLR sentiment analysis makes its decisions through supervised learning based on dictionaries of sequential behavior derived from large corpora of event data. The dictionaries and their n-gram anomaly thresholds are computed automatically and do not require the manual definition of patterns.
- SEQUIN uses unsupervised learning to extract repeating or deviating patterns from a sequence of events. This is done via grammar inference, which is not reliant on predefined patterns of any kind. It stands to mention that rule transformation going beyond the initial version introduced in Section 5.4.4 would benefit from continuous labeling. However, the respective labels need not be specified beforehand but could be assigned a designation as part of the analyst's assessment of an inferred sequence. The terminals making up the rule could even be transformed into a signature for other detection systems, making SEQUIN a feasible support and knowledge extraction system in its own right.
- Star structure anomaly detection uses templates of nominal behavior automatically selected using one of 4 implemented algorithms. This supervised approach is not reliant on patterns and computes all decision thresholds automatically. Anomaly classification uses a number of generic questions as features, which mostly avoid any kind of specifics that could be interpreted as heuristic pattern. The exception to the rule is the list of Windows function library categories: This list is likely to somewhat change with every major OS release, making it the only source of data in need of occasional updates.

Ubiquitous kernel process analysis. With the automated identification of kernel processes most expedient for observation, AIDIS is fully sample-independent in its operation. No advance knowledge of malware binaries or suspicious applications is required. The results presented in Chapter 7 show that analyzing and classifying process instances of omnipresent Windows components is very promising indeed, as almost 88.5% of all investigated attacks are reflected by events created in the first 10 seconds of a single kernel process (`svchost.exe`). This underlines the feasibility of the approach and drastically reduces the number of binaries that need to be analyzed to reach a verdict on a system's state of infection.

Transparent (white-box) anomaly detection. Each of the aforementioned anomaly detection components operate transparently. This means that the algorithms implemented for AIDIS always generate a list of events that can be seen as responsible for the anomaly alert, thereby explaining how the respective decision was reached. For

sentiment analysis, explication takes the form of event n-grams associated with an LLR score representing the maliciousness of the sequence. In SEQUIN's case, our tool yields a list of events akin to Table 8.1 as well as several pattern occurrence statistics (see Section 5.4.3). All rules computed in the inference process can be looked up in a database for maximum insight. Star structure anomaly detection creates a structured list of baseline-deviating events that can be rendered as graph.

Classification and dissemination of identified anomalies. As discussed in Section 7.4.5, the RF and SVM multi-class classification of detected (star structure) anomalies is one of the major features of AIDIS. Each threat is assigned a CAPEC attack pattern, which describes the specific adversary action independent from malware family or other sample-based categorization. The labeled anomaly, which consists of a sequence of events, can now be transformed to a format supported by the PenQuest model or turned into a behavioral signature similar to SEQUIN's output. Our results show (see Section 8.1.1) a high classification accuracy for our experiment, both in binary good/bad and multi-class scenarios.

Interpretation through comprehensive attacker–defender modeling. Few anomaly detection systems provide a full attacker–defender model for interpretative tasks (see Section 8.2.1 for a comparison of features). With AIDIS, we map the detected and classified anomalies directly to PenQuest, which offers a large repository of attack patterns and mitigating controls via established information security sources. Attack vectors, adversary goals, and system inter-dependencies are all considered. This allows analysts to explore likely actor motivations and helps locate vulnerable systems, as well as plan appropriate defensive measures to counter the threat on a technical and organizational level. Through our model, we narrow the semantic gap between data and the purpose and nature of an observed cyber-attack. AIDIS as a whole therefore provides the means to detect, classify, interpret, and disseminate all information pertaining to the possible APT, paving the way for automated threat response.

8.1.3 Research Questions and Hypotheses

Research Questions

In this section we revisit the research questions posed at the beginning of the project and discuss if and how they were answered. Afterwards, hypotheses are tested by referencing/summarizing relevant portions of research. Refer to Section 1.3 for more details about question design.

All three research questions could be answered in the course of the project. We discuss the respective answers in the following:

How can advanced targeted attacks be comprehensively modeled in preparation for semantic enrichment? In Chapter 6, we investigated gamified attacker/defender modeling as means to depict targeted attacks on arbitrary infrastructures. This follows an initial assessment of KAOS [259], GRL [319], and the i* Strategic De-

pendency (SD) model [92] as part of the literature survey, as well as the development of TAON [188], our first targeted attack ontology modeling actors, goals, and threat actions. Ultimately, TAON became PenQuest. In response to the research question, we argue that a gamified attacker/defender approach is not only capable of modeling hostile and mitigating actions on various assets, but also offers a way to automate the inference of optimal strategies leading to system compromise or successful attack prevention. This possible future research direction is discussed in Section 8.2.3 below.

How can suspicious system behavior be accurately identified without relying on predefined patterns? The drawbacks of pattern-based systems has been discussed at length in Section 1.2. The use of anomaly detection and machine learning in general offers a solution to the problem. However, the efficiency of such systems has often been disputed. With AIDIS, we show that all stages of data processing can be completed successfully without resorting to more than a bare minimum of fixed patterns: Only some of the competency questions used as features by the RF and SVM classifiers could be understood as patterns, as they refer to e.g. the existence of certain types of files within an observed anomaly. Our solution offers high accuracy and computes its results within a workable amount of time; even in comparison to other anomaly-based works (see Section 8.2.1 below).

How can identified system anomalies be mapped to specific attacks? To answer this question, we focused our research on a classification system that can be tied to a comprehensive threat model. In the top-down research stage (see Section 3.1), we designed PenQuest to support external threat information languages and attack pattern repositories such as CAPEC. An internal mapping mechanism, introduced in Section 6.5, matches these patterns to an APT kill chain phase as well as sub-categories describing their specific purpose. Target assets and potential actors are in turn connected through PenQuest's rule system. On the technical side, AIDIS Core classifies each anomaly detected by our star graph anomaly component by its preassigned CAPEC label, which serves as link to the model. Alternatively, the APT (sub-)stage itself could serve as class for supervised learning. With some adaptations, the model could also be used as malware family classifier.

In summary, targeted attacks can be accurately identified and interpreted using a combination of a granular threat modeling that supports semantically annotated attack patterns, and a classification system based on supervised machine learning, where the data assessed corresponds to an anomaly that can be mapped to said pattern.

Hypotheses

Most of the initially stated hypotheses have proven to be correct and could be confirmed through experimental research. As propounded in Chapter 2, APT detection and analysis is currently lacking, as it focuses on singular areas and rarely considers the full picture. AIDIS proves that a semantics-aware approach is better suited to learn about advanced threats and plan for their mitigation. Specifically:

The detection of APTs can be improved by using semantics-aware techniques and tools. This initial hypothesis was also addressed as part of the literature survey: The rising prominence of targeted attacks makes it necessary for new monitoring and analysis solutions to consider such scenarios from the get-go. We have shown that including both the host and network domain is vital and that the semantic classification of explained anomalies is feasible for kernel event sequences captured on an endpoint. Both SEQUIN and the PenQuest model added a formal component to the system that helps to confirm this hypothesis.

System anomalies describing attacker behavior are more feasible to use in a holistic system/network environment than fixed misuse scenarios. The overall results of our anomaly-based system confirm that classification accuracy is identical or superior to signature-based systems, such as the ones reviewed in Chapter 2. This eliminates one of the major drawbacks inherent to many behavioral systems [180]. AIDIS' successful classification of threats indirectly confirms its resistance to conventional (substitution) mimicry attacks. More advanced semantic obfuscation, adversarial learning, and feature poisoning will have to be further investigated in the context of our system.

Knowledge extraction followed by model-based attack explication is a viable approach to understanding targeted threats. AIDIS achieves knowledge extraction through its transparent explanation of anomalies. Each of the components – LLR, SEQUIN, and the star anomaly system – provides or infers events that constitute deviating behavior. This information, be it labeled as CAPEC attack pattern or semantic rule of SEQUIN, can be mapped to the PenQuest model. Experiments exhibited in Chapter 7 confirm that our take on attack explication is a promising means of closing the semantic gap, as PenQuest offers insight into adversary strategies and suggests sensible threat mitigation options.

Observing ubiquitous OS kernel processes is a feasible alternative to sample-focused analysis. Our results confirm that monitoring a single omnipresent system process is sufficient to spot nearly 88.5% of all attacks conducted on a Windows system. No prior knowledge of sample names, suspicious processes, or attack time frame is required. This arguably makes the AIDIS approach a viable substitute or extension for sample-centric analysis systems, which generally focus on individual malware binaries. It stands to mention that the SEQUIN subsystem is well suited to the task of analyzing hitherto unseen samples, regardless of our focus on ubiquitous processes.

Considering traces of abstracted behavior is more effective than observing raw API calls. Experiments conducted in Chapter 4 demonstrate that smartly reordered traces of abstracted system behavior captured by Sonar is indeed more accurate than assessing raw API calls in strict sequence of occurrence. Specifically, the difference in accuracy for the same dataset is more than 25 percentage points: Raw traces scored only 73%. See Section 4.5.1 for details.

Classifying anomalies is preferable to classifying entire traces of unknown behavior. Traces describing anomalies are significantly smaller in size than full process

logs. Considering the polynomial increase in processing time (see Figure 8.3), assessing more than a minute of process data at once was found to be unfeasible. Furthermore, using full system traces achieved a lower accuracy for both binary and multi-class classification, with the difference most pronounced when using RF: Here, multi-class accuracy was slightly below 68% as opposed to over 91% for anomaly data.

8.2 Discussion and Future Work

This final discussion section it split into three parts: A side-by-side comparison of AIDIS to the highest scored APT solutions identified during literature review, a discussion of key improvements and suggestions how to implement them in the context of an organization's infrastructure, as well as a short discourse of directions proposed for future research.

8.2.1 Comparison to Key Research

In this section we take an evaluative look at key APT research identified in the literature survey (Chapter 2). Listed and scored in Table 2.2, these works contribute most to APT detection in the context of their respective domain. We discuss commonalities, differences and synergies below. Tables 8.3 and 8.4 provide an overview of system properties in regards to the research questions/hypotheses and based on the system design checklist discussed in Chapter 2. This way we directly compare our system to the reviewed solutions and reach a verdict regarding AIDIS' singular role in future threat mitigation.

Host Solutions

As discussed in Chapter 2, Jackstraws [141] offers host-side C2 traffic identification through dynamic analysis of individual malware samples. Not unlike AIDIS, it models data flows between API calls as graph. The resulting patterns are used as templates for subsequent detection. This is also the key difference to our solution: Under the hood, Jackstraws extracts graph templates not as baseline for anomaly detection, but uses them as de-facto signatures. This results in a high number (75%) of unclassified network connections. The remainder was categorized with workable accuracy, but still scores more than 9 percentage points lower than AIDIS Core. Jackstraws and the graph matching approach underneath [350] remains a viable alternative for C2 traffic detection and will have to be further investigated for use in our prototype, as it potentially offers improved processing performance.

Malheur [314, 263] offers heuristic clustering of dynamically generated application traces of malware. The distance to a representative prototype is determined, stored, and subsequently used for the comparison of new samples to the respective cluster. Although Malheur provides its own format (MIST) for sequential traces, almost any

	AIDIS	Jacob et al. [141]	Rieck et al. [263]	Dolgikh et al. [79]	Miles et al. [213]	Ou et al. [238]
Primary domain	Host	Host	Host	Host	Host	Host
Data gathering	Dynamic	Dynamic	Dynamic	Dynamic	Static	Dynamic
Review category	-	Behavior extr.	Malware classif.	Behavior extr.	Exec. path analys.	Host-based IDS
Review score	-	8.5	8.5	7.0	7.0	7.0
Goals	Prevention Intelligence Correlation Detection Analysis	Intelligence Detection	Intelligence Detection	Intelligence Detection	Intelligence Analysis	Prediction Detection
Threat mitigation	Host intrusion	Malware	Malware	Malware	Malware	Host intrusion
Network domain	✓	✓	✓	✗	✓	✓
Analysis technique	Behavioral	Behavioral	Behavioral	Behavioral	Attribute	-
Input data	Event traces	Event traces	Event traces	Event traces	Event traces Raw/binary	Logs Network traffic Event traces
APT stages	1, 3, 4, 5, 6, 7	6	4, 5, 6, 7	4, 5, 7	3, 4, 5, 6, 7	3, 4, 5, 6, 7
Threat model	✓	✗	✗	✗	✗	✓
Pattern avoidance	✓	✗	✓	✓	✗	✗
Classification/clustering	✓	✓	✓	✗	✗	✗
Interpretation	✓	✗	✗	✗	✗	✓
Sample-independent	✓	✗	✗	✓	✗	✓
Time complexity	Polynomial	Polynomial	Linear	Polynomial [245]	Unknown	Polynomial [46]
Accuracy (binary)	99.8%	90.5%	see text	-	-	-
Accuracy (multi-class)	95.7%	-	see text	-	-	-

Table 8.3: Comparison of AIDIS to reviewed host-domain solutions referenced in Table 2.2. The given accuracy values refer to the AIDIS Core classification system; see Table 8.1 for an evaluation overview of individual AIDIS components. Note that 'threat mitigation' highlights the main focus of the system and not necessarily its full capabilities. See Chapter 2 and Section 7.3 for a complete list.

	AIDIS	Razzaq et al. [257]	Zarras et al. [352]	Ansarinia et al. [15]	Balduzzi et al. [22]	Vance [322]
Primary domain	Host	Network	Network	Network	Network	Network
Data gathering	Dynamic	Dynamic	Dynamic	Dynamic	Dynamic	Dynamic
Review category	-	Web traffic	Web traffic	Attack-specific	Web traffic	Traffic flow
Review score	-	8.5	7.5	7.0	7.0	7.0
Goals	Prevention Intelligence Correlation Detection Analysis	Intelligence Correlation Detection Analysis	Detection	Detection	Intelligence Correlation Detection	Intelligence Detection
Threat mitigation	Netw. intrusion	Netw. intrusion	Malware	Netw. intrusion	Netw. Intrusion	Netw. intrusion
Host domain support	✓	✗	✓	✓	✓	✗
Analysis technique	Behavioral	Contextual	Behavioral	Contextual	Contextual	Contextual
Input data	Event traces	Network traffic	Network traffic	-	Network traffic	Network traffic
APT stages	1, 3, 4, 5, 6, 7	1, 3, 6, 7	6, 7	1, 3, 6	1, 3, 6	6
Threat model	✓	✓	✗	✓	✗	✗
Pattern avoidance	✓	✗	✗	✗	✓	✓
Classification/clustering	✓	✗	✓	✗	✓	✓
Interpretation	✓	✓	✗	✓	✗	✗
Sample-independent	✓	✓	✗	✓	✓	✓
Time complexity	Polynomial	Linear	Unknown	-	Polynomial	Linear
Accuracy (binary)	99.8%	85.9%	99.9%	-	-	-
Accuracy (multi-class)	95.7%	-	-	-	-	-

Table 8.4: Comparison of AIDIS to reviewed network-domain solutions listed in Table 2.2. The remaining 3 solutions are considered foundational works and are discussed in text only.

event sequence generated by another analysis system can be supplied. With its focus on rapid function n-gram processing comes a lack of transparency: Prototypes or prototype distances are only expressed as numeric values. Rieck et al. assess the accuracy of the system through its F_1 score (f-measure) [249], which is defined as follows:

$$F_1 = \frac{2 * \frac{TP}{TP+FN} * \frac{TP}{TP+FP}}{\frac{TP}{TP+FN} + \frac{TP}{TP+FP}}$$

Using MIST and third-party monitoring data, Malheur achieves a clustering f-measure of 0.936 to 0.95 [263]. Classification achieves a best-case F_1 score of 0.981 for known malware samples. Note that a direct comparison to AIDIS' scores is not soundly possible because of missing sensitivity/specificity values. Still, the overall level of accuracy is estimated as within the same bracket of 99.75 to 100%. Using formats other than MIST reduces F_1 all across the board, which was confirmed by our own experiments. The authors evaluated their system with a real-world dataset of malware labeled by an anti-virus product. However, since Malheur categorizes traces by behavior instead of malware family, the separation was not cleanly reflected by the resulting clusters. Additionally, 10.7% of traces could not be assigned a prototype at all. Ultimately, we consider Malheur a valuable support system and alternative to e.g. SVM classification or similarity hashing. We decided to use Rieck et al.'s system as part of the 'prototype extraction' process of AIDIS (see Section 7.4.5), were it achieved a non-MIST F_1 score of slightly below 80%. Malheur will be evaluated as RF and SVM alternative for AIDIS' CAPEC classification in the future – should the conversion of our current graph representation into the MIST format prove to be feasible.

Dolgikh et al. [79] automatically create application profiles through graph-based function/parameter tracing of malware or benign system processes. API calls are transformed into a labeled graph representing a stream of events. Graphs are then compressed using the Graphitour algorithm [245], resulting in rules that, in case of benign applications, represent nominal behavior. This information can then be used as part of an anomaly detection system. While the approach is generally similar to the SEQUIN grammar inference component of AIDIS, a direct comparison is difficult because of the system's lack of comprehensive evaluation.

With VirusBattle, Miles et al. [213] focus on the similarity of malware instances to discover links between actors, machines, and malware: Code, semantically similar procedures of code, and API call execution/event log traces are compared to identify similarities. This includes visited websites, e-mail messages, and static PE file headers. VirusBattle can best be compared to AIDIS' Sonar monitoring agent with the key difference being its reliance on static analysis through concolic execution. Miles et al.'s approach connects various levels of information to infer new semantic links. In contrast, AIDIS always maintains process context and does not need to reconnect the dots in the first place – a distinct advantage of dynamic analysis. User information and machine IDs are readily available in the database as well. Furthermore, we can link observed events

to likely actor classes and even attack descriptions. The VirusBattle paper does not provide the numbers necessary for a side-by-side accuracy evaluation of the approach.

Ou et al. [238] present a formal model depicting the (human) reasoning process used to determine whether a system is compromised. Unlike AIDIS, they primarily use captured network traffic scanned by an existing IDS configured with pre-existing intrusion alert patterns. The reasoning engine converts signature descriptions to rules, where individual events contribute to the overall certainty level of compromise. In short, Ou et al.'s system interprets its findings in accordance to existing threat descriptions with all the shortcomings of a signature-reliant approach, while AIDIS bases its classification on the outcome of a supervised learning process using anomalies. Formally, Ou et al. [238] use a Datalog-like language [46], whereas PenQuest provides detailed class definitions and a comprehensive 'game manual' for the representation of its ruleset.

Network Solutions

The reviewed research by Razzaq et al. [257] presents a detection and classification system for web application attacks through an ontological model. The model includes semantic rules for validating protocol formats and inferring malicious activity. At its core, however, the system still relies on fixed rules (regular expressions) describing various attacks, even if it promises higher accuracy than purely sequential network-based IDS. On average, it detected 18 classes of evaluated network attacks with a mean accuracy of 85.9%, whereby SQL injections were detected most reliably (95.4%) and information leakage attacks posed the greatest challenge (78.1% accuracy). Similar to AIDIS, the system by Razzaq et al. attaches a short description of consequence to its rules (e.g. 'accessing sensitive information'). Our own model assigns a patterns class ('IL–illegal access') describing attacker behavior. Each class is then associated with specific (CAPEC) attack patterns and their properties. While AIDIS/PenQuest arguably offer more detailed semantic information and even mitigation strategies, it does not yet boast the same level of automation as Razzaq et al.'s system.

With its focus on HTTP traffic generated by malware, Zarras et al.'s BotHound [352] primarily offers template-based detection of C2 traffic. The system uses an optimized threshold similar to the star graph anomaly detection component (sans classification) of AIDIS. Similar in scope to our 23+2 CAPEC classes, BotHound was trained with traffic of a total of 24 malware families. However, all classification done by Zarras et al. only assigned a benign or malicious label; no separation into malware families was performed. In the binary scenario, the system's accuracy was comparable to AIDIS' SVM-based classification (see Figure 8.6).

In [15] the authors create an attack ontology based on threat intelligence information gleaned from CAPEC, CWE, and CVE. In addition to similar data sources, their work can be compared to PenQuest's foundation ontology, TAON [188]. While Ansarinia et al.' system aims to construct an ontology for describing DDoS-related pat-

terns, it does not attempt to map real-world data to the model. With no concrete input data, accuracy comparisons cannot be performed.

SpuNge [22] focuses on behavioral clustering of URL strings in order to correlate attacks targeting a specific industry or location. This is achieved mostly through string similarity measurement, after which similar request/response information is grouped together. For example, binary files transmitted over the network that use a similar file name will likely be put in the same cluster. Like SEQUIN, SpuNge is primarily used for data reduction and pattern inference, as it helps identify event commonalities such as shared C2 servers. It was not evaluated as automated detection system by its authors.

Vance [322]'s work introduces a flow-based traffic monitoring system capable of statistical anomaly detection. The system uses change detection through sketch-based measurement [165] to identify flows that hint at command & control traffic, data mining, or exfiltration activities. It can be understood as a support solution that primarily reduces the amount of flow data in need of processing by highlighting the changes. This, in a way, makes it similar to AIDIS' classification system, where we consider anomalies instead of entire event traces. Vance claims to have achieved an increase in APT alerts with his approach, but fails to provide specific numbers for a direct comparison.

Other Solutions

The remaining three works listed in Table 2.2 are discussed below. As they represent general research and modeling efforts, a quantitative comparison (table) was omitted.

With Gestalt, Atighetchi et al. [21] introduce a multi-source data management system for centralized forensic data query and analysis. All information is abstracted using a custom language. Gestalt is best compared to the database server hosting our agent's collected monitoring data. They key difference to AIDIS' storage and querying system is that Gestalt leaves the (log, traffic, alert) data where it was generated. Semantic queries are translated to low-level requests to e.g. a database server. With Sonar, we use raw SQL queries to extract the data needed for anomaly detection from a central system. Since the design of the agent and its database server is not within the scope of this thesis, a direct comparison was omitted. However, any future AIDIS implementation will undoubtedly benefit from changes emulating Gestalt's take on data collection and processing. For more discussion on this topic, see Section 8.2.2.

Gorodetski et al. [116] sketch a threat modeling and traffic simulation system for multi-source data similar to Gestalt's. Semantic information such as attacker intentions are considered, which makes Gorodetski et al.'s model something of an early cousin to AIDIS/PenQuest and SIEM systems in general.

With ADeLe, Totel et al. [312] present a language specifically tailored to describe exploits and attacks from the target's perspective. In the context of AIDIS, the component most similar to Totel et al.'s research is the mapping mechanism linking identified anomalies to the PenQuest model (see Section 7.4.5), which is currently realized through

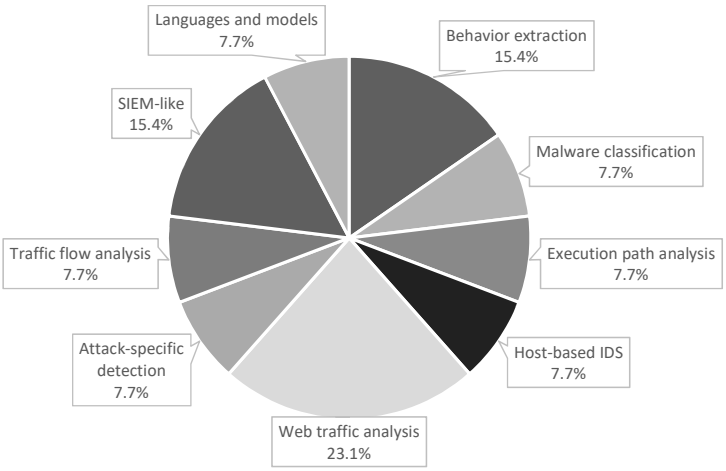


Figure 8.4: Distribution of solution categories found in all of the compared systems.

time-enhanced event sequences exemplified in 7.5.3. ADeLe additionally supports recurring events, which are not currently distinguished by AIDIS. Refer to 8.2.2 below for planned future improvements to the event-to-model mapping process.

Summary

With 5 host, 5 network and 3 general solutions, this side-by-side comparison took a closer look at the differences and commonalities of existing research and our AIDIS approach. The key findings are summarized here. Please note that the final 3 works could not be quantified and are therefore omitted from most of the below statistics.

Of the 10 remaining works all but one use or necessitate dynamic data gathering for collecting their input. Almost all of them offer threat detection, closely followed by threat intelligence. Correlation, analysis, and prediction are rarely seen. Response and prevention is not covered by the third-party systems at all. By comparison, AIDIS offers all of the above with the exception of automated response and prediction. Such functionality is discussed in Section 8.2.3, where we take a detailed look at future research directions.

As shown in Figure 8.4, behavior extraction and web traffic analysis are the most represented categories. If classified accordingly, AIDIS would best fit into the host-based IDS cluster, followed by behavior extraction and malware classification. Behavioral and contextual analysis techniques are each used in 4 of the systems. One solution uses attribute-based analysis. AIDIS itself primarily uses behavioral analysis.

Out of the 10 quantified solutions, 5 focus on malware detection – 4 of which require a pre-identified sample to operate. One system is close to a host-based IDS similar to what AIDIS would be classified as, while the remainder (4) is designed to deal with network intrusions. Our approach remains the only one able to cover all APT attack stages, save off-site weaponization. With 9 systems in total, kill chain stage 6 (C2 communication) is considered the most.

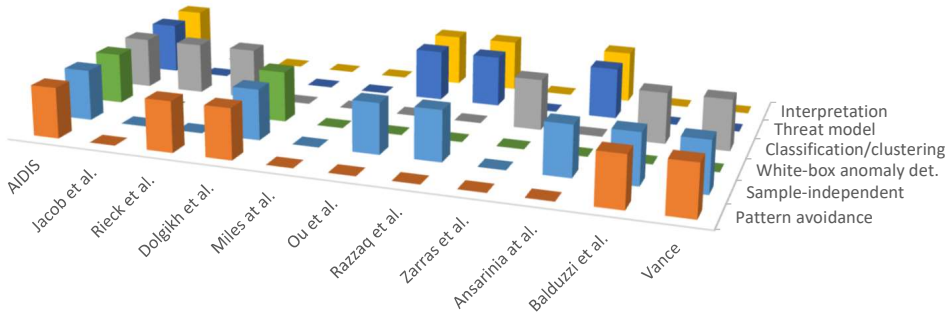


Figure 8.5: Overview of AIDIS features present in other solutions. Contribution 5 (interpretation through comprehensive attacker–defender modeling) has been split up into 'threat model' and 'interpretation' for this comparison.

As discussed in Section 8.1.2, there are five aspects we have identified as key contributions of AIDIS, most of which have been initially defined as part of the research questions and hypotheses. Since AIDIS is a specialized multipurpose solution it was unlikely to find other tools with exactly the same capabilities. Indeed, only 5 works manage to provide three of the five aforementioned aspects. The remainder share even less of AIDIS' distinguishing features. Sample-independence is represented the most (6), closely followed by clustering/classification capabilities. Threat models and interpretative features are rarely seen; such are typically found in formal solutions that do not offer a link to real-world data. Anomaly detection, if supported at all, fails to provide transparent decision-making in all but one cases. A complete feature overview can be found in Figure 8.5.

Lastly, we take a closer look at classification accuracy of systems providing this information. Only the solution by Zarras et al. [352] promises scores similar to AIDIS. Their 'BotHound' prototype achieved 99.9% accuracy in binary scenarios. Multiple classes are not supported. The system also relies on samples, patterns, and does not utilize a threat model or interpretation components. The works by Jacob et al. [141] and Razzaq et al. [257] range between 85 and 91% benign/malicious classification accuracy. The only system on the shortlist supporting multi-class assignment is Malheur [263]: F_1 scores achieved by that system are discussed separately above.

Despite the limited number of systems AIDIS can be compared to, it is apparent that most research identified during the early stages of the project does not match its capabilities in terms of functionality and accuracy. The system proposed in this thesis should offer information security specialists novel means of combating attacks on their infrastructure. To better prepare our prototype for productive implementation we discuss specific areas of improvement below.

8.2.2 Limitations and Improvements

In this section we discuss the various areas of improvement as well as future work identified in previous chapters and sketch how a productive implementation of AIDIS would

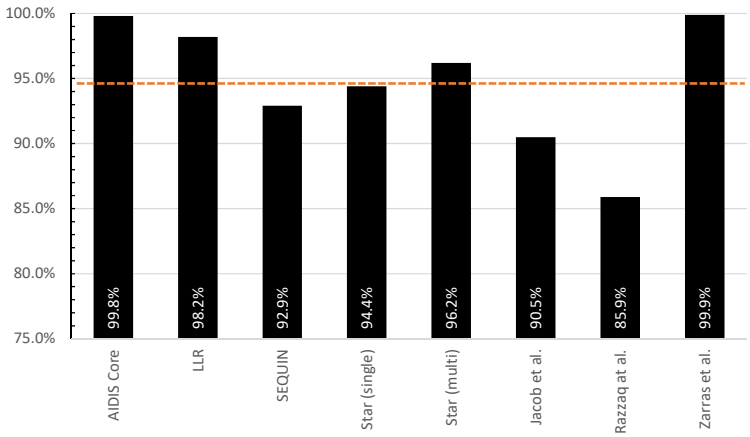


Figure 8.6: Binary classification accuracy comparison of the AIDIS core system, its individual components (best values), as well as similar solutions identified during literature review. The dotted line represents the overall average.

look like in a real-world setting. Focusing on the functionality and modus operandi of a hypothetical system beyond its current prototype stadium, we create a list of future research tasks and discuss desired system features. Four areas are considered in particular: The data basis and possible changes thereof, information processing and analysis automation, overall performance improvements, as well as steps to boost detection and classification accuracy.

Data

In the following, we investigate possible changes to the data selection and research instruments as well as useful new features in regards to a wider OS compatibility.

Expanded data selection. There are several areas where data captured by an updated Sonar agent or even an additional external data provider (see Section 7.4.1) may lead to better results and richer threat semantics. In isolated experiments [182], we assessed the feasibility of adding NetFlow data to complement kernel-level network events. NetFlow is a network traffic monitoring solution originally developed by Cisco [57]. In addition to exiting source/destination addresses and port information, it offers TCP flags, IP header and routing information, as well as packet and byte counts. With its possible deployment directly on routers and switches, NetFlow offers supplementary observation points outside the local OS. This data would enable AIDIS to build baseline templates based on information such as the average number of bytes transferred by each process or the presence of specific flags within a dataframe. Since NetFlow does not provide PID/TID information, correlation has to be achieved by matching timestamps and destination IP addresses to the respective arguments contained in the corresponding kernel calls.

Locally, it makes sense to investigate additional types of events that could be captured by the existing kernel monitoring agent. That includes e.g. certificate events, which produce a data point whenever the local certificate store is altered, as well as the

use of encryption functions possibly relevant for ransomware detection. Furthermore, logging user context outside of filesystem or registry paths would be a valuable addition for detecting impersonation or privilege escalation attacks.

Operating system compatibility. Currently, the Sonar monitoring agent only supports API call hooking inside the Microsoft Windows ecosystem. Database, preprocessing and all analysis stages have been coded with Windows 7+ events in mind. In order to provide Unix/Linux compatibility, several changes to the system are necessary: First and foremost, a new agent needs to be developed from scratch. Alternatively, tools like strace[1], ptrace[2], or the audit subsystem[3] could be utilized to collect the required behavioral information. Next, the database structure has to be updated to reflect the changes in event handling and to support additional event types. Abstraction routines and preprocessing scripts require changes as well – mainly due to differences in the Unix filesystem and its directory structure. Because of their largely OS-neutral implementation, the actual AIDIS components can be amended for Unix environments with no to little effort: LLR and SEQUIN are fully compatible out of the box. Star structures and the RF/SVM classification system will have to be updated to include new event types and competency questions; the lack of dynamic link libraries and registry events has also to be considered when leaving the Windows platform.

New evaluation instruments. On a non-technical side, the currently employed data instruments for the evaluation of PenQuest – namely the feedback questionnaire – will benefit from some structural changes. In its first iteration, the model has been tested as gamified risk assessment and awareness tool for use in higher education. For AIDIS Core, random sample testing determined if the suggested countermeasures following the classification and mapping stages are sensible (see Chapter 7.5.3). However, a full expert evaluation of PenQuest's output is still outstanding. For this purpose, new testing scenarios and a new questionnaire need to be prepared. We plan to present a group of security professionals with a list of several anomaly reports and their corresponding threat/mitigation annotations as provided by PenQuest. The participants will be asked to grade in detail the system's suggestions in order to ascertain practical applicability. Future work will also include customized network maps for threat assessments tailored to specific organizations. Like before, this process will be subject to human review. See 'Automated data to model mapping' below for additional improvements to the gamified model.

Processing and Automation

Processing and automation is undoubtedly a vital aspect of AIDIS. The complexity of our approach invites the discussion of alternative means of data processing, data mapping automation, and additional knowledge discovery routines in future implementations.

[1] https://strace.io/
[2] https://linux.die.net/man/2/ptrace
[3] https://linux.die.net/man/8/auditd

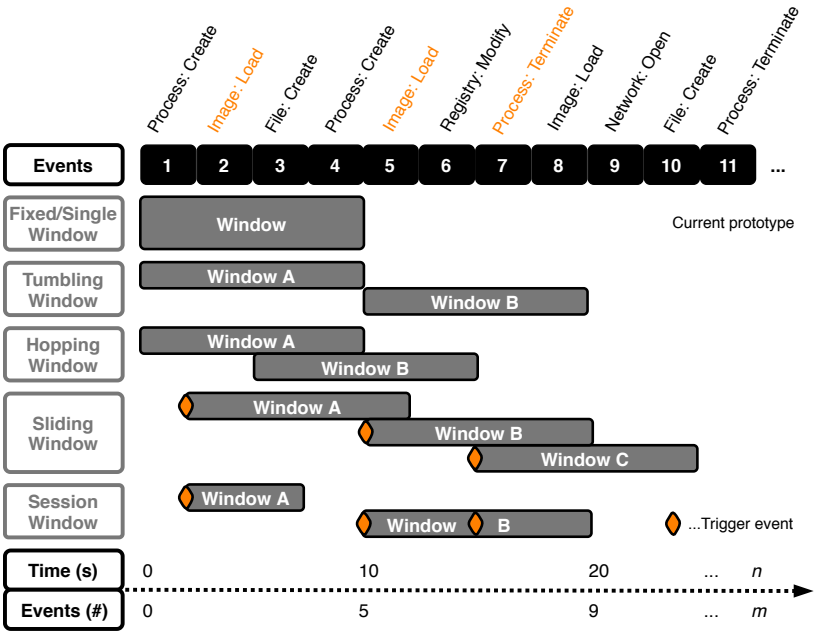


Figure 8.7: Overview of window processing options in conjecture with event streaming. New windows are either determined by time or by the occurrence of an event. For standard session windowing, the presence of at least m anomalous events within a time period $t = y_w$ extends the window W by another y_w seconds. Here: $m = 1$, $y_w = 2$.

Stream window processing. In AIDIS' current prototype, all data processing is either conducted on full system traces depicting the entire runtime of a process (e.g. as part of the LLR component), or using a fixed window describing n seconds of activity containing possible star structure anomalies. In both cases, the respective dictionary or template of an unknown trace is compared against the same temporal frame: For example, an unclassified 10-second trace is always compared against a template describing the same 10-second period of its baseline process. While this works well in our experiments, there are alternative means of processing that need to be considered for a production version of AIDIS. The reasons are manifold: With processes less generic than `svchost.exe`, attackers may delay their hostile activity until after the window of analysis, thereby evading detection. Additionally, the complexity of computation will not always allow full trace processing but may require the use of streaming, where a window of inspection is moved over the data in a specific manner. Streaming also implies that the data is not statically retrieved from the database but continuously processed as it arrives. As shown in Figure 8.7, there are several approaches to stream windowing that need to be considered for future versions of our system. We follow the Azure Stream Analytics nomenclature [208] for discussing the various alternatives.

There are two general approaches to stream-based processing: *time-* or *count-based windowing*, and *event-triggered windowing*. For temporal windows, the time passed decides when to stop analysis or when to move the window to its next position. Alternatively, AIDIS allows the use of event counts instead of a fixed time range: Here, the window is shifted only after e.g. 300 events have been recorded instead of after 10 seconds of process runtime. This is useful for apps that include longer wait times or where

a temporal window may contain too many events for efficient computation. Figure 8.3 provides sensible thresholds.

Event-triggered windowing such as the 'sliding window' and 'session window' approach (see Figure 8.7) are designed to move the inspection range whenever a certain event is registered. The latter method is even capable of grouping and dynamically extending the window depending on whether they occur within a certain timeout period.

Since AIDIS' star graph component is not using pattern matching on strictly temporal data, we also need to consider template selection. A certain window of inspection does not necessarily have to imply the use of a same-length, same-time template. Even a 'hopping window' with a small shift of e.g. 1 second will struggle with 'template drift', the fact that any additional event registered in an anomalous trace might push benign events semantically belonging to the same window out of the inspection range. Additionally, purely temporal or count-based windowing will cause a lot of computational overhead with data that does not contain a large number of deviating events. For this reason, we suggest a custom 'session window' approach for future implementations:

First, a full star template T for the entire process runtime needs to be created using one of AIDIS' single- or multi-template methods. This has to be repeated at intervals short enough to prevent concept drift (i.e. weeks to months) or whenever major changes are made to the setup of the observed machines. This template serves as decision maker for creating a new session window: Whenever the stream delivers an event $e_{trigger}$ not contained in the full template, a session window W is created. It is dynamically extended by a count of events $e_{trigger} + x_w$ or a number of seconds $t_{trigger} + y_w$ whenever another anomalous event is registered. This keeps the session window growing until no further triggers are observed, or until a maximum window size $W_{max} = a * x_w$ or $W_{max} = b * y_w$ has been reached, whereas a and b are limited by available computational power. Once a session 'ends', the actual anomaly detection process will start, computing the graph edit distance to a reduced template T_{part} for a suggested interval of $\{e - x_w, \ldots, e + 2x_w\}$ or $\{t - y_w, \ldots, t + 2y_w\}$, respectively. This ensures that any activity happening close to the trigger event will be considered even if we are facing noticeable template drift.

Summed up, the conversion to stream-based processing is recommended for IDS scenarios that require continuous monitoring instead of application launch or full process behavioral analysis as currently conducted by AIDIS. Further experiments are required to prove the effectiveness of the suggested session window approach and its aforementioned alternatives.

Automated data to model mapping. As of the time of writing, the PenQuest model exists as set of axioms, class definitions, and rules prototypically implemented as physical strategy game. The mapping of detected and classified anomalies happens mostly manually. To automate this process, future work is planned to include the conversion of anomaly data from the basic $\langle Event \rangle$ format to established threat intelligence languages such as STIX: Individual events could be translated to STIX Cyber Observable

Objects[1], corresponding to various file system activity. For example, the initial process launch that is part of the sequence shown in Section 7.4.5 (Table 6.8) might be denoted as follows:

```
{
  "0": {
    "type": "file",
    "hashes": {
      "SHA-256": "9f86d081884c7d659a2feaa0c55ad01 (...)"
    },
  },
  "1": {
    "type": "process",
    "name": "parent.exe",
    "created": "2019-01-01T16:00:000Z",
  },
  "2": {
    "type": "process",
    "name": "shell.exe",
    "created": "2019-01-01T16:53:661Z",
    "binary_ref": "0"
    "parent_ref": "1"
  }
}
```

As de-facto standard for describing threat intelligence [274], STIX includes a wide range of observables that are generally similar to the events supported by our system. The "parent_ref" property even allows for the representation of (partial) process trees as constructed by our monitoring agent during preprocessing (exemplified above). There is still work to be done, however: New properties will have to be introduced to encompass the full range of details provided by AIDIS and Sonar. PenQuest's semantic annotations will require additional adaptation of existing STIX objects, likely the ones used to describe attack patterns, campaigns, and action relationships[2]. Integrating our system into the data models of languages such as STIX or MISP[3] will generally help to improve threat information sharing between organizations.

An alternative approach to the structured description of events linked to the model would be to use a formal language akin to ADeLe [312]. For instance, the $\langle Event \rangle$ sequence referenced above could be denoted as follows:

```
<ENCHAIN>
    Sequence(process-shell.exe_START_process-drop.exe,
             process-drop.exe_LOAD_image-library.dll,
             process-drop.exe_OPEN_registry-HKLM/Software/.../Run,
             process-drop.exe_ADD_registry-REG_SZ(evil.exe))
</ENCHAIN>
```

ADeLe supports user context and the specification of minimum and maximum temporal delay between individual events, which would provide a sensible addition to PenQuest's $\langle Event \rangle$ class. While specific third-party research needs to be surveyed in more detail to prepare our model for automation, reasoning and storage, it is also

[1] http://docs.oasis-open.org/cti/stix/v2.0/cs01/part4-cyber-observable-objects/stix-v2.0-cs01-part4-cyber-observable-objects.html

[2] http://docs.oasis-open.org/cti/stix/v2.0/cs01/part2-stix-objects/stix-v2.0-cs01-part2-stix-objects.html

[3] https://www.misp-project.org/

reasonable to revisit our own TAON ontology [188], which has been designed as both predecessor and possible extension to the PenQuest system. We will investigate building an updated Protégé ontology to support the new classes and to provide a uniform OWL XML interface for linking events to TAON's hierarchical storage. To exemplify the process, below snippet depicts an Individual of the $\langle Victim \rangle$ class as defined in Section 6.3.2, written here in Manchester OWL syntax:

```
Individual: Victim_1_DatabaseServer
   Types:
     Asset
   Facts:
     hasParent Victim_2_WebServer,
     hasParent Disabler_1_IDS,
     hasVectorParent Victim_2_WebServer,
     hasVectorParent Victim_7_MobileDevice,
     hasConfiguration Tech_NIST_SI-3_MaliciousCodeProtection,
     hasConfiguration Org_NIST_AC-2_AccountManagement,
     (...)
     containsKnowledge Information_9_CustomerData,
     containsKnowledge Configuration_4_HashedPasswords,
     (...)
     Exposure "exposed",
     Status "running",
     Integrity "affected",
     Name "Generic SQL Database Server"
```

With the remodeling of PenQuest's classes and axioms into an ontology format it becomes possible to provide others with shareable repositories of threats, (mitigating) configurations, and sequences of events registered on the system itself – be that in an ADeLe-like language, STIX, or the internal OWL XML format. Summed up, the next stages of research will revolve around ontology construction and the conversion of classified anomaly reports into a compatible format that can be automatically parsed and stored.

Additional knowledge discovery and prediction. As a semantics-aware solution designed to infer or provide (anomalous) event information at each step, AIDIS offers invaluable insight into attacker behavior. Future work will encompass a more in-depth analysis of preferred tools and actions, in addition to automated knowledge discovery using KAMAS and other (visual) analysis systems. For example, the output of both LLR and the SEQUIN component can be used to collect information about malicious domains potentially involved in C2 communication. By extracting addresses from network events it becomes trivial to build new blacklists for conventional IDS solutions. LLR additionally provides the means to infer event types or directory names that are more likely to lead to a compromise.

Armed with the respective anomaly scores we can build predictive models that assess the chance of future misbehavior – a huge step towards a more proactive defense. As a first step, we plan to update the competency questions currently used for RF and SVM classification and apply them to the event trace dataset in its entirety. The goal is to compare binary classification to our current anomalies-only approach and to test the system's suitability for regression analysis. Similar efforts are planned for SEQUIN, which is designed to express nominal behavior through its induction of grammar rules.

Future research into visual analytics will be conducted to help experts spot and label deviations. See Section 5.7.2 and [332] for more information.

Performance

As discussed in Section 8.1.1, performance remains an issue with some components of the system. Below, we consider future optimization in terms of code improvements and data generalization.

Code optimization. The re-implementation of some of the prototypes developed as part of the project promises significant performance boosts across the board. While the main LLR and Kuhn-Munkres computations have been realized mainly in R, the assembly of the matrices used for these components is in particular need of improvement: Currently, most of the necessary parsing/restructuring work is done using Linux on-board tools, which do not support parallel processing and execute an excess number of I/O operations when reading trace files. A full R implementation or new code based on Python's SciPy[1] suite are under investigation. In-database computation will also be considered as part of our efforts to move to a stream-based format that supports session windowing.

Data generalization. In our experiment design (see Section 3.2) we opted to perform as little data generalization as possible to prevent the loss of information potentially required for later interpretation. Nevertheless, AIDIS offers mechanisms to abstract directory paths and registry hive names to semantic descriptors; for example, a full registry path pointing at the configuration settings of a system app could be generalized to `\LocalMachine\Software`. Likewise, an arbitrary temporary directory within the user's `AppData` folder might just be labeled as such. While this undoubtedly leads to a loss of positional information, future experiments will have to determine the feasibility of such an approach – namely how much information can be omitted before data generalization reduces classification accuracy.

Detection and Classification Accuracy

Lastly, we take a closer look at improvements that could potentially increase the overall accuracy of the system. Here, we focus on changes to the LLR component, general feature optimization of AIDIS Core, as well as on the topic of data labeling in general.

Trigram processing. In their study, Lanzi et al. [176] determined that n-grams of length 3 provided the best results for their AccessMiner system. Using bigrams and 4-grams resulted in significantly lower accuracy scores. Our own experiments with a random set of kernel events could not corroborate these findings, but where too small in scope to be truly representative. A full side-by-side evaluation of variable-length n-grams is still pending. It stands to mention that the bigram classification accuracy of our LLR standalone system was markedly higher than Lanzi et al. [176]'s implementation,

[1] https://www.scipy.org/

however. Future work will detail the proposed experiment, while at the same time increase the amount of data used to evaluate the LLR system.

Feature optimization. The selection of features is key for any detection system's classification accuracy. As discussed in Section 7.5.3 and depicted in Figure 7.9, not all of the features assessed in our evaluation are of identical import. Further optimizing and automating the creation of competency questions will undoubtedly improve overall results. Additionally, adding the score and verdict of the other AIDIS components to the feature set needs to be investigated: Only the analysis result of the star anomaly system is currently among the features used in the prototype. The reason why including LLR and SEQUIN was not a priority to date can be found in the relatively minor contribution of the respective star structure feature: In terms of mean decrease in accuracy, it cannot be found in the top 30.

Data labeling challenge. One of the biggest issues with any kind of classification system is the quality of labeled data. In our experiments we used real-world monitoring information captured from machines used by software developers and office personnel. While this provides a behavioral snapshot close to the reality of deployment, it still comes with a semantic bias towards the IT sector and might not result in data representative for other businesses. Similarly, the label quality of any malicious software observed by the Sonar agent depends on the insight and know-how of the analyst who originally dissected the sample. Any errors during labeling will skew the results generated by AIDIS' RF and SVM classifier. In order to address this formidable challenge, further efforts need to be made to create shared sets of endpoint event monitoring data similar to the testing repositories available for network traffic analysis [2, 65].

Last but not least, future work will exploit the support of select machine learning algorithms for assigning multiple labels. It is not guaranteed that even the 10-second 'fixed window' evaluated as part of AIDIS' classification contains activity associated with only a single (CAPEC) attack pattern. On the contrary, it is highly likely that a malicious actor will perform a range of actions that need to be mapped to different $\langle AttackClass \rangle$ items. Multiple labels are expected to synergize well with session windowing, where the determined window can serve as a semantic boundary to the pattern in question. Algorithms supporting multi-label classification include, but are not limited to: ML-k-nearest neighbor (kNN) [355], MH-AdaBoost [123], decision tree variants [324], and stream-based approaches such as ADWIN Bagging [258].

8.2.3 Future Research Directions

Next to above improvements to the AIDIS system, we see the continued development of the PenQuest gamified model as key to improving APT mitigation and defense planning. While the model can already be used for that purpose on a case-by-case basis there is a distinct lack of automation in regards to the computation of optimal defense strategies that apply to a given scenario. For this purpose, future research may focus on formal analysis through model checking or on e.g. a reinforcement learning (RL) approach for

inferring a defender's ideal response. Generally speaking, we suggest to evaluate our game-based approach by utilizing deep learning methods that aim to maximize the 'reward' (speed, stealth, effectiveness) in the context of our gamified model.

An inference system based on PenQuest would likely focus on answering the following general questions:

- Which attack strategies can be considered most dangerous in terms of systemic effect?
- Which access vectors to an arbitrary IT infrastructure are most vulnerable to hostile action?
- Which combinations of attack patterns facilitate the fastest, stealthiest, or most impactful compromise of a system?
- In turn, which defensive measures and controls are able to best reduce the success rate and impact of specific attacks in terms of delay, detection, or alleviation?

In the following, we outline two possible approaches to achieving the goal of automated strategy inference.

Model Checking

Model checking [59] is an established technique which complements the traditional (penetration) testing approach for validating ICT systems. While testing only explores one specific execution of a given system, the idea of model checking is to systematically explore all possible runs in search for system states that may violate a required property. If a state violating the specified property (e.g. system or data confidentiality, integrity, or availability) is found, an execution of the system leading to the error state is generated. In practice, model checking suffers from what is known as the state space explosion problem [183]: The reachable state space grows exponentially with the size of the system. For this reason, explicitly exploring the state space is not feasible. An alternative is to combine the strength of model checking with the practicality of a traditional testing approach: Model-based testing techniques [77] automatically generate test cases based on exploring a (simplified) system model. These techniques are especially useful for achieving test coverage and for generating test cases which will stress potentially weak parts of the system.

As part of future research such an approach could be used to enable the analysis of a given IT infrastructure with regard to its vulnerability – respectively resilience – to APTs. The goal would be to come up with modeling techniques and algorithms which allow the analyst to automatically answer queries like the following:

- How can I violate confidentiality, integrity, or availability [299] of a specific level of impact?
- Given an attack vector X, how can I best defend?
- What are likely attack vectors?

- What are the minimal requirements an attack actor needs to meet in order to have a certain chance of successfully violating confidentiality, integrity, or availability of the given infrastructure?
- Generally speaking, which kind of likely attack actors are the most threatening to our infrastructure?

While model checking may not be the answer to all the general questions posed above, it promises the ability to automate significant portions of a future inference process.

Reinforcement Learning

Next to supervised and unsupervised learning, reinforcement learning [151] constitutes the third group of machine learning methods. The basic idea behind reinforcement learning is that agents interacting with their environment receive rewards or penalties by repeatedly performing actions that aim to achieve certain goals. This procedure is carried out until a final system state is reached. The agent has to learn the optimal strategy – the policy – by itself. The goal of the agent is to explore the state-action space in order to find the best strategy by maximizing the possible reward. To achieve this goal, algorithms like Temporal Difference (TD) learning [302], Q-Learning [335], or Deep Q-Learning [226] are used to calculate individual states and actions in advance. This explorational ability of the learning algorithm empowers agents to find more successful strategies than human operators might [225, 239, 325].

Any such reinforcement learning system could be able to compute optimal strategies that promise the highest probability of success when it comes to attacking or defending an infrastructure. Tying in to the sketched model checking approach, another goal would be to help maximizing/minimizing confidentiality, integrity, or availability impact on the assessed infrastructure. Other inference runs would focus on detection chance, the time required for successfully conducting/mitigating an attack before major harm can be done, or the optimization of monetary cost.

8.3 Résumé

This thesis presented AIDIS, a full-fledged threat detection and interpretation system capable of detecting and classifying anomalous behavior in abstracted OS software activity. We specifically focused on omnipresent system processes that are part of the Windows kernel and are therefore ideal candidates for continuous, sample-independent monitoring. AIDIS's premise is to keep anomaly detection transparent to the user, thereby allowing analysts to check how an individual score was computed. Events responsible for an alert are always disclosed. Based on this information, we successfully mapped the resulting reports to PenQuest, our own attacker–defender model capable of illustrating adversarial activity as actions in a strategy game. Through the model, all

captured and classified data can be assigned a tendency towards a specific APT stage and technical attack pattern as defined by external vocabularies. At the same time, PenQuest provides asset and actor details and suggests specific controls intended to counter an observed attack. This versatility makes AIDIS one of the first truly semantics-aware threat detection systems to combine applied analysis of real-world data with model-aided interpretation of anomalies.

The system's many features are expected to help industry professionals in their task of discovering, assessing, and mitigating threats on their infrastructures. Thanks to our flexible model and the white-box approach to anomaly classification, possible deployment scenarios are not limited to the domain of conventional IT and might in the future cover industrial scenarios, physical security, and even human behavior.

As a next step, AIDIS will likely be implemented as part of CyberTrap, an upcoming security solution by an Austrian company of the same name, which seeks to spot and analyze attacker behavior on a high-interaction honeypot cloned from a production system. The SEQUIN component is continuously developed as unsupervised pattern inference system for industrial settings. PenQuest, in its game iteration, is expected to be released as educational app in 2020.

With its semantic take on endpoint intrusion detection and classification, AIDIS offers a holistic solution to an ever-growing threat. The white-box approach helps analysts understand adversary behavior at the lowest levels, while the PenQuest model incorporates threat intelligence and risk management. We hope that, over time, our work will help to prove true the famous words of Sun Tzu:

"If you know the enemy and know yourself,
you need not fear the result of a hundred battles."

Bibliography

[1] Fateme Abdoli and Mohsen Kahani. Ontology-based distributed intrusion detection system. In *Computer Conference, 2009. CSICC 2009. 14th International CSI*, pages 65–70. IEEE, 2009.

[2] Adamu I Abubakar, Haruna Chiroma, Sanah Abdullahi Muaz, and Libabatu Baballe Ila. A review of the advances in cyber security benchmark datasets for evaluating data-driven based intrusion detection systems. *Procedia Computer Science*, 62:221–227, 2015.

[3] Pieter W Adriaans et al. *Learning shallow context-free languages under simple distributions*. Institute for Logic, Language and Computation (ILLC), University of Amsterdam, 1999.

[4] Alfred V Aho, Ravi Sethi, and Jeffrey D Ullman. *Compilers, Principles, Techniques*. Addison Wesley, 1986.

[5] Wolfgang Aigner, Daniel A Keim, Sebastian Schrittwieser, Robert Luh, Fabian Fischer, Alexander Rind, Dominik Sacha, and Markus Wagner. Visual analytics: Foundations and experiences in malware analysis. In *Empirical Research for Software Security*, pages 159–192. CRC Press, 2017.

[6] Manoun Alazab, Robert Layton, Sitalakshmi Venkataraman, and Paul Watters. Malware detection based on structural and behavioural features of API calls. 2010. URL http://ro.ecu.edu.au/icr/1/.

[7] Ahmed AlEroud and George Karabatis. A system for cyber attack detection using contextual semantics. In *7th International Conference on Knowledge Management in Organizations: Service and Cloud Computing*, pages 431–442. Springer, 2013.

[8] Ahmed Aleroud and George Karabatis. Context Infusion in Semantic Link Networks to Detect Cyber-attacks: A Flow-Based Detection Approach. pages 175–182. IEEE, June 2014. ISBN 978-1-4799-4003-5 978-1-4799-4002-8. doi: 10.1109/ICSC.2014.29.

[9] Alienvault. OSSIM: The Open Source SIEM | AlienVault, 2015. URL https://www.alienvault.com/products/ossim.

[10] T. Ambwani. Multi class support vector machine implementation to intrusion detection. In *Proceedings of the International Joint Conference on Neural Networks, 2003.*, volume 3, pages 2300–2305 vol.3, July 2003. doi: 10.1109/IJCNN.2003.1223770.

[11] Theodoros Anagnostopoulos, Christos Anagnostopoulos, and Stathes Hadjiefthymiades. Enabling attack behavior prediction in ubiquitous environments. In *Pervasive Services, 2005. ICPS'05. Proceedings. International Conference on*, pages 425–428. IEEE, 2005.

[12] Blake Anderson, Daniel Quist, Joshua Neil, Curtis Storlie, and Terran Lane. Graph-based malware detection using dynamic analysis. *Journal in computer Virology*, 7(4):247–258, 2011.

[13] Stig Andersson, Andrew Clark, George Mohay, Bradley Schatz, and Jacob Zimmermann. A framework for detecting network-based code injection attacks targeting Windows and UNIX. In *Computer Security Applications Conference, 21st Annual*, pages 10–pp. IEEE, 2005.

[14] Krasimir Angelov. Incremental parsing with parallel multiple context-free grammars. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, pages 69–76. Association for Computational Linguistics, 2009.

[15] Morteza Ansarinia, Seyyed Amir Asghari, Afshin Souzani, and Ahmadreza Ghaznavi. Ontology-based modeling of DDoS attacks for attack plan detection. In *Telecommunications (IST), 2012 Sixth International Symposium on*, pages 993–998. IEEE, 2012.

[16] Apache Software Foundation. Apache JENA, 2015. URL `https://jena.apache.org/`.

[17] Gorka Irazoqui Apecechea, Mehmet Sinan Inci, Thomas Eisenbarth, and Berk Sunar. Fine grain Cross-VM Attacks on Xen and VMware are possible! *IACR Cryptology ePrint Archive*, page 248, 2014.

[18] Alberto Apostolico and Concettina Guerra. The longest common subsequence problem revisited. *Algorithmica*, 2(1-4):315–336, 1987.

[19] Keven Ates and Kang Zhang. Constructing veggie: Machine learning for context-sensitive graph grammars. In *Tools with Artificial Intelligence, 2007. ICTAI 2007. 19th IEEE International Conference on*, volume 2, pages 456–463. IEEE, 2007.

[20] Keven Ates, Jacek Kukluk, Lawrence Holder, Diane Cook, and Kang Zhang. Graph grammar induction on structural data for visual programming. In *Tools with Artificial Intelligence, 2006. ICTAI'06. 18th IEEE International Conference on*, pages 232–242. IEEE, 2006.

[21] Michael Atighetchi, John Griffith, Ian Emmons, David Mankins, and Richard Guidorizzi. Federated Access to Cyber Observables for Detection of Targeted Attacks. pages 60–66. IEEE, October 2014. ISBN 978-1-4799-6770-4. doi: 10.1109/MILCOM.2014.15.

[22] Marco Balduzzi, Vincenzo Ciangaglini, and Robert McArdle. Targeted attacks detection with spunge. In *Privacy, Security and Trust (PST), 2013 Eleventh Annual International Conference on*, pages 185–194. IEEE, 2013.

[23] Sean Barnum. Standardizing cyber threat intelligence information with the Structured Threat Information eXpression (STIX™). *MITRE Corporation*, 11:1–22, 2012.

[24] Ulrich Bayer, Christopher Kruegel, and Engin Kirda. TTAnalyze: A tool for analyzing malware. *EICAR*, pages 180–192, 2006.

[25] Ulrich Bayer, Paolo Milani Comparetti, Clemens Hlauschek, Christopher Kruegel, and Engin Kirda. Scalable, Behavior-Based Malware Clustering. In *NDSS*, volume 9, pages 8–11. Citeseer, 2009.

[26] BBN Technologies. Asio BBN, 2015. URL http://asio.bbn.com.

[27] Kristian Beckers and Sebastian Pape. A serious game for eliciting social engineering security requirements. In *Requirements Engineering Conference (RE), 2016 IEEE 24th International*, pages 16–25. IEEE, 2016.

[28] Dmitri Bekerman, Bracha Shapira, Lior Rokach, and Ariel Bar. Unknown malware detection using network traffic classification. In *Communications and Network Security (CNS), 2015 IEEE Conference on*, pages 134–142. IEEE, 2015.

[29] Fabrice Bellard. Qemu, a fast and portable dynamic translator. In *USENIX Annual Technical Conference, FREENIX Track*, volume 41, page 46, 2005.

[30] Kenneth Benoit, Kohei Watanabe, Paul Nutly, Adam Obeng, Haiyan Wang, Benjamin Lauderdale, and Will Lowe. *Quanteda: Quantitative Analysis of Textual Data*, 2017. URL http://quanteda.io.

[31] Frank Benteler. Layout-Graphgrammatiken für die Darstellung von hierarchisch strukturierten Graphen am Beispiel von Wellendigitalstrukturen. 2002.

[32] Sandeep Bhatkar, Abhishek Chaturvedi, and R. Sekar. Dataflow anomaly detection. In *Security and Privacy, 2006 IEEE Symposium on*, pages 15–pp. IEEE, 2006.

[33] Parth Bhatt, Edgar Toshiro Yano, and Per Gustavsson. Towards a Framework to Detect Multi-stage Advanced Persistent Threats Attacks. pages 390–395. IEEE, April 2014. ISBN 978-1-4799-3616-8. doi: 10.1109/SOSE.2014.53.

[34] Leyla Bilge and Tudor Dumitras. Before we knew it: an empirical study of zero-day attacks in the real world. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 833–844. ACM, 2012.

[35] Bob Blakley, Ellen McDermott, and Dan Geer. Information security is information risk management. In *Proceedings of the 2001 workshop on New security paradigms*, pages 97–104. ACM, 2001.

[36] Michael D. Bond and Kathryn S. McKinley. Probabilistic calling context. In *ACM SIGPLAN Notices*, volume 42, pages 97–112. ACM, 2007.

[37] Michael D. Bond, Varun Srivastava, Kathryn S. McKinley, and Vitaly Shmatikov. Efficient, context-sensitive detection of real-world semantic attacks. In *Proceedings of the 5th ACM SIGPLAN Workshop on Programming Languages and Analysis for Security*, page 1. ACM, 2010.

[38] Ed Bott. Why malware networks are beating antivirus software, 2015. URL http://www.zdnet.com/article/why-malware-networks-are-beating-antivirus-software/.

[39] Edward J Briscoe. Language as a complex adaptive system: co-evolution of language and of the language acquisition device. In *Proceedings of Eighth Computational Linguistics in the Netherlands Conference*, 1998.

[40] Andrei Z Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences 1997. Proceedings*, pages 21–29. IEEE, 1997.

[41] BSI. Durchführung von Planspielen zur Informationssicherheit. Technical report, 2014.

[42] João Paulo da Costa Calado. *Open source IDS/IPS in a production environment: comparing, assessing and implementing*. PhD thesis, 2018.

[43] Sergio Caltagirone, Andrew Pendergast, and Christopher Betz. The diamond model of intrusion analysis. Technical report, Center for Cyber Intelligence Analysis and Threat Research, Hanover, 2013.

[44] Brian Caswell and Jay Beale. *Snort Intrusion Detection 2.0*. Syngress, May 2003. ISBN 978-0-08-048100-5.

[45] William B. Cavnar, John M. Trenkle, and others. N-gram-based text categorization. *Ann Arbor MI*, pages 161–175, 1994.

[46] Stefano Ceri, Georg Gottlob, and Letizia Tanca. What you always wanted to know about Datalog (and never dared to ask). *IEEE transactions on knowledge and data engineering*, 1(1):146–166, 1989.

[47] Abhishek Chaturvedi, Sandeep Bhatkar, and R. Sekar. Improving attack detection in host-based IDS by learning properties of system call arguments. In *In Proceedings of the IEEE Symposium on Security and Privacy*. Citeseer, 2005.

[48] M. Chen, D. Ebert, H. Hagen, R.S. Laramee, R. Van Liere, K.-L. Ma, W. Ribarsky, G. Scheuermann, and D. Silver. Data, information, and knowledge in visualization. *Computer Graphics & Applications*, 29(1):12–19, 2009. ISSN 0272-1716. doi: 10.1109/MCG.2009.6.

[49] Raymond Chen. When does a process ID become available for reuse?, 2019. URL https://blogs.msdn.microsoft.com/oldnewthing/20110107-00/?p=11803/.

[50] Hsiu-Sen Chiang and Woei-Jiunn Tsaur. Ontology-based Mobile Malware Behavioral Analysis. *Da-Yeh University*, 2009.

[51] Eric Chien, Liam OMurchu, and Nicolas Falliere. W32. Duqu: the precursor to the next stuxnet. In *Proceedings of the 5th USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2012.

[52] Sheng-Hui Chien, Erh-Hsien Chang, Chih-Yung Yu, and Cheng-Seen Ho. Attack subplan-based attack scenario correlation. In *Machine Learning and Cybernetics, 2007 International Conference on*, volume 4, pages 1881–1887. IEEE, 2007.

[53] Mihai Christodorescu, Somesh Jha, Sanjit Seshia, Dawn Song, Randal E. Bryant, and others. Semantics-aware malware detection. In *Security and Privacy, 2005 IEEE Symposium on*, pages 32–46. IEEE, 2005.

[54] Mihai Christodorescu, Somesh Jha, and Christopher Kruegel. Mining specifications of malicious behavior. In *Proceedings of the 1st India software engineering conference*, pages 5–14. ACM, 2008.

[55] Cisco. Cisco IOS NetFlow, 2015. URL http://cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html.

[56] Cisco. Snort.Org, 2015. URL https://www.snort.org/.

[57] Benoit Claise. Cisco systems NetFlow services export version 9. Technical report, 2004.

[58] Alexander Clark. Unsupervised induction of stochastic context-free grammars using distributional clustering. In *Proceedings of the 2001 workshop on Computational Natural Language Learning-Volume 7*, page 13. Association for Computational Linguistics, 2001.

[59] Edmund M Clarke, Orna Grumberg, and Doron Peled. *Model checking*. MIT Press, 1999.

[60] C. Collins, F.B. Viegas, and M. Wattenberg. Parallel tag clouds to explore and analyze faceted text corpora. In *Symp. on Visual Analytics Science and Technology*, pages 91–98, 2009. doi: 10.1109/VAST.2009.5333443.

[61] Alain Colmerauer and Philippe Roussel. The birth of Prolog. In *History of programming languages—II*, pages 331–367. ACM, 1996.

[62] Charles Consel and Olivier Danvy. Static and dynamic semantics processing. In *Proceedings of the 18th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 14–24. ACM, 1991.

[63] Allan Cook, Richard Smith, Leandros Maglaras, and Helge Janicke. Measuring the risk of cyber attack in industrial control systems. BCS eWiC, 2016.

[64] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[65] Gideon Creech and Jiankun Hu. Generation of a new IDS test dataset: Time to retire the KDD collection. In *Wireless Communications and Networking Conference (WCNC), 2013 IEEE*, pages 4487–4492. IEEE, 2013.

[66] Gideon Creech and Jiankun Hu. A Semantic Approach to Host-Based Intrusion Detection Systems Using Contiguousand Discontiguous System Call Patterns. *Computers, IEEE Transactions on*, 63(4):807–819, 2014. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6419701.

[67] Cuckoo Foundation. Automated Malware Analysis - Cuckoo Sandbox, 2015. URL http://www.cuckoosandbox.org/.

[68] Michel Cukier. Study: Hackers attack every 39 seconds. URL https://eng.umd.edu/news/story/study-hackers-attack-every-39-seconds.

[69] Mila Dalla Preda, Mihai Christodorescu, Somesh Jha, and Saumya Debray. A semantics-based approach to malware detection. *ACM Transactions on Programming Languages and Systems*, 30(5):1–54, 2008.

[70] Santanu Das, Bryan L Matthews, Ashok N Srivastava, and Nikunj C Oza. Multiple kernel learning for heterogeneous anomaly detection: algorithm and aviation safety case study. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 47–56. ACM, 2010.

[71] Jelle De Vries, Hans Hoogstraaten, Jan van den Berg, and Semir Daskapan. Systems for Detecting Advanced Persistent Threats: A Development Roadmap Using Intelligent Data Analysis. In *Intl. Conference on Cyber Security*, pages 54–61. IEEE, 2012.

[72] Herve Debar and Andreas Wespi. Aggregation and correlation of intrusion-detection alerts. In *Recent Advances in Intrusion Detection*, pages 85–103. Springer, 2001.

[73] Herve Debar, David Curry, and Benjamin Feinstein. The Intrusion Detection Message Exchange Format (IDMEF), 2015. URL https://www.ietf.org/rfc/rfc4765.txt.

[74] Hervé Déjean. ALLiS: a symbolic learning system for natural language learning. In *Proceedings of the 2nd workshop on Learning language in logic and the 4th*

conference on Computational natural language learning-Volume 7, pages 95–98. Association for Computational Linguistics, 2000.

[75] Dell SecureWorks. Truman, 2015. URL http://www.secureworks.com/cyber-threat-intelligence/tools/truman/.

[76] Jaime Devesa, Igor Santos, Xabier Cantero, Yoseba K. Penya, and Pablo Garcia Bringas. Automatic Behaviour-based Analysis and Classification System for Malware Detection. In *ICEIS (2)*, pages 395–399, 2010.

[77] Arilo C Dias Neto, Rajesh Subramanyan, Marlon Vieira, and Guilherme H Travassos. A survey on model-based testing approaches: a systematic review. In *Proceedings of the 1st ACM international workshop on Empirical assessment of software engineering languages and technologies: held in conjunction with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE) 2007*, pages 31–36. ACM, 2007.

[78] Darina Dicheva, Christo Dichev, Gennady Agre, and Galia Angelova. Gamification in education: a systematic mapping study. *Journal of Educational Technology & Society*, 18(3):75, 2015.

[79] Andrey Dolgikh, Tomas Nykodym, Victor Skormin, and Zachary Birnbaum. Using behavioral modeling and customized normalcy profiles as protection against targeted cyber-attacks. In *Computer Network Security*, pages 191–202. Springer, 2012.

[80] Hermann Dornhackl, Konstantin Kadletz, Robert Luh, and Paul Tavolato. Malicious behavior patterns. In *2014 IEEE 8th International Symposium on Service Oriented System Engineering (SOSE)*, pages 384–389. IEEE, 2014.

[81] Hermann Dornhackl, Konstantin Kadletz, Robert Luh, and Paul Tavolato. Defining malicious behavior. In *9th International Conference on Availability Reliability and Security (ARES)*, pages 273–278. IEEE, 2014.

[82] Jair Moura Duarte, João Bosco dos Santos, and Leonardo Cunha Melo. Comparison of similarity coefficients based on RAPD markers in the common bean. *Genetics and Molecular Biology*, 22(3):427–432, 1999.

[83] Thomas E. Dube, Richard A. Raines, Michael R. Grimaila, Kenneth W. Bauer, and Steven K. Rogers. Malware Target Recognition of Unknown Threats. *IEEE Systems Journal*, 7(3):467–477, September 2013. ISSN 1932-8184, 1937-9234. doi: 10.1109/JSYST.2012.2221913.

[84] Ted Dunning. Accurate methods for the statistics of surprise and coincidence. *Computational linguistics*, pages 61–74, 1993.

[85] Ted Dunning. Surprise and Coincidence - musings from the long tail, 2008. URL http://tdunning.blogspot.co.at/2008/03/surprise-and-coincidence.html.

[86] Arianna D'Ulizia, Fernando Ferri, and Patrizia Grifoni. A survey of grammatical inference methods for natural language learning. *Artificial Intelligence Review*, 36 (1):1–27, 2011.

[87] Kenneth S Edge, George C Dalton, Richard A Raines, and Robert F Mills. Using attack and protection trees to analyze threats and defenses to homeland security. In *Military Communications Conference, 2006. MILCOM 2006. IEEE*, pages 1–7. IEEE, 2006.

[88] Bradley Efron and Robert J Tibshirani. *An Introduction to the Bootstrap*. CRC press, 1994.

[89] Manuel Egele, Theodoor Scholte, Engin Kirda, and Christopher Kruegel. A survey on automated dynamic malware-analysis techniques and tools. *ACM Computing Surveys*, 2012. URL http://dl.acm.org/citation.cfm?id=2089126.

[90] E.E. Eiland, S.C. Evans, T.S. Markham, and J.D. Impson. MDL compress system and method for signature inference and masquerade intrusion detection, December 4 2012. URL https://www.google.com/patents/US8327443. US Patent 8,327,443.

[91] Patrick Henry Engebretson, Joshua J Pauli, and Kevin Streff. Abstracting parent mitigations from the capec attack pattern dictionary. In *Security and Management*, pages 245–250, 2008.

[92] S. Yu Eric. Social Modeling and i*. In *Conceptual Modeling: Foundations and Applications*, pages 99–121. Springer, 2009.

[93] Javier Esparza, Martin Leucker, and Maximilian Schlund. Learning workflow petri nets. In *International Conference on Applications and Theory of Petri Nets*, pages 206–225. Springer, 2010.

[94] Andrea Esuli and Fabrizio Sebastiani. SentiWordNet: A publicly available lexical resource for opinion mining. In *Proceedings of LREC*, volume 6, pages 417–422. Citeseer, 2006.

[95] Nicolas Falliere, Liam Murchu, and Eric Chien. W32.Stuxnet.Dossier, 2015. URL https://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf.

[96] James A. Farrell, 2015. URL http://www.cs.man.ac.uk/~pjj/farrell/comp2.html.

[97] Stefan Fenz and Andreas Ekelhart. Formalizing information security knowledge. In *Proceedings of the 4th international Symposium on information, Computer, and Communications Security*, pages 183–194. ACM, 2009.

[98] Mariano Fernández-López, Asunción Gómez-Pérez, and Natalia Juristo. Methontology: from ontological art towards ontological engineering. 1997.

[99] Eric Filiol, Grégoire Jacob, and Mickaël Le Liard. Evaluation methodology and theoretical model for antiviral behavioural detection strategies. *Journal in Computer Virology*, 3(1):23–37, 2007.

[100] FireEye. FireEye Malware Analysis, 2015. URL https://www.fireeye.com/products/malware-analysis.html.

[101] International Organization for Standardization. *ISO 5725-1: 1994: Accuracy (Trueness and Precision) of Measurement Methods and Results-Part 1: General Principles and Definitions*. ISO Geneva, Switzerland, 1994.

[102] Stephanie Forrest, Steven A Hofmeyr, Anil Somayaji, and Thomas A Longstaff. A sense of self for unix processes. In *IEEE Symposium on Security and Privacy*, pages 120–128. IEEE, 1996.

[103] The Open Information Security Foundation. Suricata open source IDS / IPS / NSM engine, 2019. URL https://suricata-ids.org/.

[104] Mark S. Fox, Mihai Barbuceanu, and Michael Gruninger. An organisation ontology for enterprise modelling: preliminary concepts for linking structure and behaviour. In *Enabling Technologies: Infrastructure for Collaborative Enterprises, 1995., Proceedings of the Fourth Workshop on*, pages 71–81. IEEE, 1995.

[105] M. Franz. Dynamic linking of software components. *Computer*, 30(3):74–81, March 1997. ISSN 0018-9162. doi: 10.1109/2.573670.

[106] Matt Fredrikson, Somesh Jha, Mihai Christodorescu, Reiner Sailer, and Xifeng Yan. Synthesizing near-optimal malware specifications from suspicious behaviors. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 45–60. IEEE, 2010.

[107] Thomas Freytag. Woped–Workflow Petri Net designer. *University of Cooperative Education*, pages 279–282, 2005.

[108] Yasuhiro Fukushima, Akihiko Sakai, Yoichi Hori, and Kouichi Sakurai. A behavior based malware detection scheme for avoiding false positive. In *Secure Network Protocols (NPSec), 2010 6th IEEE Workshop on*, pages 79–84. IEEE, 2010.

[109] Roland Gabriel, Tobias Hoppe, Alexander Pastwa, and Sebastian Sowa. Analyzing Malware Log Data to Support Security Information and Event Management: Some Research Results. pages 108–113. IEEE, 2009. ISBN 978-1-4244-3467-1. doi: 10.1109/DBKDA.2009.26.

[110] Thomas Gamer, M. Scholler, and Roland Bless. A granularity-adaptive system for in-network attack detection. In *Proceedings of the IEEE/IST Workshop on Monitoring, Attack Detection and Mitigation*, pages 47–50, 2006.

[111] Michael Gamon. Sentiment classification on customer feedback data: noisy data, large feature vectors, and the role of linguistic analysis. In *Proceedings of the 20th*

international conference on Computational Linguistics, page 841. Association for Computational Linguistics, 2004.

[112] Chandan Gautam, Ramesh Balaji, K Sudharsan, Aruna Tiwari, and Kapil Ahuja. Localized multiple kernel learning for anomaly detection: One-class classification. *Knowledge-Based Systems*, 165:241–252, 2019.

[113] Paul Giura and Wei Wang. A context-based detection framework for advanced persistent threats. In *Cyber Security (CyberSecurity), 2012 International Conference on*, pages 69–74. IEEE, 2012.

[114] GlobalSecurity.org. Open Source Information System (OSIS), 2015. URL http://www.globalsecurity.org/intell/systems/osis.htm.

[115] Robert P Goldberg. Survey of virtual machine research. *Computer*, (6):34–45, 1974.

[116] Vladimir Gorodetski, Igor Kotenko, and Oleg Karsaev. Multi-agent technologies for computer network security: Attack simulation, intrusion detection and intrusion detection learning. *Comput. Syst. Sci. Eng.*, 18(4):191–200, 2003.

[117] David Gotz, Harry Stavropoulos, Jimeng Sun, and Fei Wang. ICDA: A platform for intelligent care delivery analytics. *AMIA Annual Symp. Proceedings*, 2012:264–273, 2012. ISSN 1942-597X. URL http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3540495/.

[118] Adam Greenberg. Russians fingered for 'Uroburos' spy malware campaign, went undetected for years - SC Magazine, 2015. URL http://www.scmagazine.com/russians-fingered-for-uroburos-spy-malware-campaign-went-undetected-for-years/article/336570/.

[119] Peter D Grünwald. MDL tutorial. *Advances in minimum description length: Theory and applications*, 2005.

[120] André RA Grégio, Dario S. Fernandes Filho, Vitor M. Afonso, Rafael DC Santos, Mario Jino, and Paulo L. de Geus. Behavioral analysis of malicious code through network traffic and system call monitoring. In *SPIE Defense, Security, and Sensing*. Intl. Society for Optics and Photonics, 2011.

[121] Nicola Guarino and Christopher A. Welty. An overview of OntoClean. In *Handbook on ontologies*, pages 201–220. Springer, 2009.

[122] Michael D Hanus and Jesse Fox. Assessing the effects of gamification in the classroom: A longitudinal study on intrinsic motivation, social comparison, satisfaction, effort, and academic performance. *Computers & Education*, 80:152–161, 2015.

[123] Trevor Hastie, Saharon Rosset, Ji Zhu, and Hui Zou. Multi-class AdaBoost. *Statistics and its Interface*, 2(3):349–360, 2009.

[124] Vasileios Hatzivassiloglou and Kathleen R. McKeown. Predicting the semantic orientation of adjectives. In *Proceedings of the 35th annual meeting of the association for computational linguistics*, pages 174–181. Association for Computational Linguistics, 1997. URL http://dl.acm.org/citation.cfm?id=979640.

[125] Peng He and George Karabatis. Using semantic networks to counter cyber threats. In *Intelligence and Security Informatics (ISI), 2012 IEEE International Conference on*, pages 184–184. IEEE, 2012.

[126] Liona Herman. Malware Attack at US Health Organization Went Undetected for 2 Years, 2015. URL http://www.hackbusters.com/news/stories/187232-malware-attack-at-us-health-organization-went-undetected-for-2-years.

[127] Hex-Rays. IDA: About, 2015. URL https://www.hex-rays.com/products/ida/.

[128] Soshi Hirono, Yukiko Yamaguchi, Hajime Shimada, and Hiroki Takakura. Development of a Secure Traffic Analysis System to Trace Malicious Activities on Internal Networks. pages 305–310. IEEE, July 2014. ISBN 978-1-4799-3575-8. doi: 10.1109/COMPSAC.2014.41.

[129] Daniel Hoffman and Richard Snodgrass. Trace specifications: Methodology and models. *IEEE Transactions on Software Engineering*, 14(9):1243–1252, 1988.

[130] Ake J Holmgren, Erik Jenelius, and Jonas Westin. Evaluating strategies for defending electric power networks against antagonistic attacks. *IEEE Transactions on Power Systems*, 22(1):76–84, 2007.

[131] Xin Hu, Tzi-cker Chiueh, and Kang G Shin. Large-scale malware indexing using function-call graphs. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 611–620. ACM, 2009.

[132] Hsien-Der Huang, Tsung-Yen Chuang, Yi-Lang Tsai, and Chang-Shing Lee. Ontology-based intelligent system for malware behavioral analysis. In *Fuzzy Systems (FUZZ), 2010 IEEE International Conference on*, pages 1–6. IEEE, 2010.

[133] Eric M. Hutchins, Michael J. Cloppert, and Rohan M. Amin. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *Leading Issues in Information Warfare & Security Research*, 1:80, 2011.

[134] Nwokedi Idika and Aditya P. Mathur. A survey of malware detection techniques. *Purdue University*, 48, 2007. URL http://cyberunited.com/wp-content/uploads/2013/03/A-Survey-of-Malware-Detection-Techniques.pdf.

[135] Piotr Indyk and Rajeev Motwani. Approximate Nearest Neighbors: Towards Removing the. 1999.

[136] Ponemon Institute. Cost of cyber crime study: Insights on the security investments that make a difference. 2017.

[137] iSecLab. Anubis, 2015. URL https://anubis.iseclab.org/.

[138] Paul Jaccard. Étude comparative de la distribution florale dans une portion des alpes et des jura. *Bull Soc Vaudoise Sci Nat*, 37:547–579, 1901.

[139] Grégoire Jacob, Hervé Debar, and Eric Filiol. Behavioral detection of malware: from a survey towards an established taxonomy. *Journal in computer Virology*, 4 (3):251–266, 2008.

[140] Grégoire Jacob, Hervé Debar, and Eric Filiol. Malware behavioral detection by attribute-automata using abstraction from platform and language. In *International Workshop on Recent Advances in Intrusion Detection*, pages 81–100. Springer, 2009.

[141] Gregoire Jacob, Ralf Hund, Christopher Kruegel, and Thorsten Holz. Jackstraws: Picking command and control connections from bot traffic. In *USENIX Security Symposium*, volume 2011. San Francisco, CA, USA, 2011.

[142] K. Jensen and Lars M. Kristensen. *Coloured Petri Nets: Modelling and validation of concurrent systems*. Springer, Dordrecht ; New York, 2009. ISBN 978-3-642-00283-0 978-3-642-00284-7.

[143] Xuxian Jiang, Xinyuan Wang, and Dongyan Xu. Stealthy malware detection through vmm-based out-of-the-box semantic view reconstruction. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 128–138. ACM, 2007.

[144] Joe Security. Agile Malware Analysis - Joe Sandbox Desktop, 2015. URL http://www.joesecurity.org/joe-sandbox-desktop.

[145] Jeff Johnson. *Designing with the Mind in Mind: Simple Guide to Understanding User Interface Design Guidelines*. Morgan Kaufmann, 2014. ISBN 978-0-12-407914-4.

[146] Joint Task Force Transformation Initiative. SP 800-53 rev. 4. Recommended Security Controls for Federal Information Systems and Organizations. Technical report, Gaithersburg, MD, United States, 2015.

[147] Seong-Wook Joo and Rama Chellappa. Attribute grammar-based event recognition and anomaly detection. In *Computer Vision and Pattern Recognition Workshop, 2006. CVPRW'06. Conference on*, pages 107–107. IEEE, 2006.

[148] Klaus Julisch and Christopher Kruegel. *Detection of Intrusions and Malware, and Vulnerability Assessment: Second International Conference, DIMVA 2005, Vienna, Austria, July 7-8, 2005, Proceedings*. Springer Science & Business Media, June 2005. ISBN 978-3-540-26613-6.

[149] Aivo Jürgenson and Jan Willemson. Serial model for attack tree computations. In *International Conference on Information Security and Cryptology*, pages 118–128. Springer, 2009.

[150] Peyman Kabiri and Ali A. Ghorbani. Research on Intrusion Detection and Response: A Survey. *IJ Network Security*, 1(2):84–102, 2005.

[151] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[152] Kaspersky Lab. Duqu: Steal Everything, 2015. URL http://www.kaspersky.com/about/press/major_malware_outbreaks/duqu.

[153] Kaspersky Lab. What is Flame Malware | Definition and Risks | Kaspersky Lab, 2015. URL http://www.kaspersky.com/flame.

[154] Kaspersky Lab's Global Research & Analysis Team. Gauss: Abnormal Distribution - Securelist, 2015. URL https://securelist.com/analysis/36620/gauss-abnormal-distribution/.

[155] D.A. Keim. Designing pixel-oriented visualization techniques: theory and applications. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):59–78, 2000. ISSN 1077-2626. doi: 10.1109/2945.841121.

[156] Kris Kendall and Chad McMillan. Practical malware analysis. In *Black Hat Conference, USA*, 2007.

[157] Richard Kissel. Glossary of key information security terms. *NIST Interagency Reports NIST IR*, 7298(3), 2013.

[158] Barbara A. Kitchenham. Procedures for undertaking systematic reviews. Technical report, Computer Science Department, Keele University, 2004.

[159] Shawn Knight. Sophisticated malware dubbed 'The Mask' went undetected for the past seven years - TechSpot, 2015. URL http://www.techspot.com/news/55640-sophisticated-malware-dubbed-the-mask-went-undetected-for-the-past-seven-years.html.

[160] Loren Kohnfelder and Praerit Garg. The threats to our products. *Microsoft Interface, Microsoft Corporation*, 1999.

[161] Barbara Kordy, Sjouke Mauw, Saša Radomirović, and Patrick Schweitzer. Foundations of attack–defense trees. In *International Workshop on Formal Aspects in Security and Trust*, pages 80–95. Springer, 2010.

[162] Barbara Kordy, Sjouke Mauw, Saša Radomirović, and Patrick Schweitzer. Attack–defense trees. *Journal of Logic and Computation*, 24(1):55–87, 2014.

[163] Bart Kosko. Fuzzy cognitive maps. *International Journal of man-machine studies*, 24(1):65–75, 1986.

[164] Igor Kotenko and Andrey Chechulin. Common Framework for Attack Modeling and Security Evaluation in SIEM Systems. pages 94–101. IEEE, November 2012. ISBN 978-1-4673-5146-1 978-0-7695-4865-4. doi: 10.1109/GreenCom.2012.24.

[165] Balachander Krishnamurthy, Subhabrata Sen, Yin Zhang, and Yan Chen. Sketch-based change detection: methods, evaluation, and applications. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pages 234–247. ACM, 2003.

[166] Christopher Kruegel, Richard Lippmann, and Andrew Clark, editors. *Recent advances in intrusion detection: 10th intl. symposium, RAID 2007, Gold Coast [i.e. Coast], Australia, September 5-7, 2007: proceedings*. Number 4637 in Lecture notes in computer science. Springer-Verlag, Berlin ; New York, 2007. ISBN 978-3-540-74319-4.

[167] Harold W Kuhn. The Hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[168] Harold William Kuhn. *Lectures on the Theory of Games*. Princeton University Press, 2009.

[169] Adam Kujawa. You dirty RAT! Part 1: DarkComet, 2012. URL https://blog.malwarebytes.com/threat-analysis/2012/06/you-dirty-rat-part-1-darkcomet/.

[170] Sandeep Kumar and Eugene H. Spafford. A pattern matching model for misuse intrusion detection. 1994.

[171] Jonghoon Kwon and Heejo Lee. Bingraph: Discovering mutant malware using hierarchical semantic signatures. In *7th Intl. Conference on Malicious and Unwanted Software*, pages 104–111. IEEE, 2012. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6461015.

[172] Arun Lakhotia, Mila Dalla Preda, and Roberto Giacobazzi. Fast location of similar code fragments using semantic'juice'. In *Proceedings of the 2nd ACM SIGPLAN Program Protection and Reverse Engineering Workshop*, page 5. ACM, 2013.

[173] Douglas J Landoll and Douglas Landoll. *The security risk assessment handbook: A complete guide for performing security risk assessments*. CRC Press, 2005.

[174] Carl E. Landwehr, Alan R. Bull, John P. McDermott, and William S. Choi. A taxonomy of computer program security flaws. *ACM Computing Surveys (CSUR)*, 26(3):211–254, 1994.

[175] Stefan Langeder. *Towards Dynamic Attack Recognition for SIEM*. PhD thesis, St. Pölten University of Applied Sciences, 2014.

[176] Andrea Lanzi, Davide Balzarotti, Christopher Kruegel, Mihai Christodorescu, and Engin Kirda. AccessMiner: Using system-centric models for malware protection.

In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 399–412. ACM, 2010.

[177] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.

[178] Ted G Lewis. *Critical infrastructure protection in homeland security: Defending a networked nation*. John Wiley & Sons, 2014.

[179] Frankie Li, Anthony Lai, and Ddl Ddl. Evidence of Advanced Persistent Threat: A case study of malware for political espionage. In *Malicious and Unwanted Software (MALWARE), 2011 6th International Conference on*, pages 102–109. IEEE, 2011.

[180] Hung-Jen Liao, Chun-Hung Richard Lin, Ying-Chih Lin, and Kuang-Yuan Tung. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1):16–24, 2013.

[181] Andy Liaw, Matthew Wiener, et al. Classification and regression by Random Forest. *R news*, 2(3):18–22, 2002.

[182] Martin Liebl. Correlating system events to support the network analysis process. Master's thesis, St. Pölten University of Applied Sciences, 2014. URL https://phaidra.fhstp.ac.at/detail_object/o:2345.

[183] Fuchun J Lin, PM Chu, and Ming T Liu. Protocol verification using reachability analysis: the state space explosion problem and relief strategies. In *ACM SIGCOMM Computer Communication Review*, volume 17, pages 126–135. ACM, 1987.

[184] Aristid Lindenmayer. Mathematical models for cellular interactions in development i. filaments with one-sided inputs. *Journal of theoretical biology*, 18(3): 280–299, 1968.

[185] Maria B. Line, Ali Zand, Gianluca Stringhini, and Richard Kemmerer. Targeted Attacks against Industrial Control Systems: Is the Power Industry Prepared? pages 13–22. ACM Press, 2014. ISBN 978-1-4503-3154-8. doi: 10.1145/2667190.2667192.

[186] John Locke. *Some thoughts concerning education*. University Press, 1895.

[187] Robert Luh, Stefan Marschalek, Manfred Kaiser, Helge Janicke, and Sebastian Schrittwieser. Semantics-aware detection of targeted attacks: a survey. *Journal of Computer Virology and Hacking Techniques*, pages 1–39, 2016.

[188] Robert Luh, Sebastian Schrittwieser, and Stefan Marschalek. TAON: An ontology-based approach to mitigating targeted attacks. In *Proceedings of the 18th International Conference on Information Integration and Web-based Applications & Services*. ACM, 2016.

[189] Robert Luh, Gregor Schramm, Markus Wagner, and Sebastian Schrittwieser. Sequitur-based inference and analysis framework for malicious system behavior. In *Proceedings of the 3rd International Conference on Information Systems Security and Privacy (ICISSP 2017)*, pages 632–643, 2017. ISBN 978-989-758-209-7. doi: 10.5220/0006250206320643.

[190] Robert Luh, Sebastian Schrittwieser, and Stefan Marschalek. LLR-based sentiment analysis for kernel event sequences. In *Advanced Information Networking and Applications (AINA), 2017 IEEE 31st International Conference on*, pages 764–771. IEEE, 2017.

[191] Robert Luh, Sebastian Schrittwieser, Stefan Marschalek, Helge Janicke, and Edgar Weippl. Design of an anomaly-based threat detection & explication system. In *ICISSP*, pages 397–402, 2017.

[192] Robert Luh, Sebastian Schrittwieser, Stefan Marschalek, Helge Janicke, and Edgar Weippl. Poster: Design of an anomaly-based threat detection & explication system. In *Proceedings of the 22nd ACM on Symposium on Access Control Models and Technologies*, pages 119–120. ACM, 2017.

[193] Robert Luh, Gregor Schramm, Markus Wagner, Helge Janicke, and Sebastian Schrittwieser. SEQUIN: a grammar inference framework for analyzing malicious system behavior. *Journal of Computer Virology and Hacking Techniques*, pages 1–21, 2018.

[194] Robert Luh, Marlies Temper, Simon Tjoa, and Sebastian Schrittwieser. APT RPG: Design of a gamified attacker/defender meta model. In *Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP 2018)*. SCITEPRESS, 2018.

[195] Robert Luh, Marlies Temper, Simon Tjoa, Sebastian Schrittwieser, and Helge Janicke. PenQuest: A gamified attacker/defender meta model for cyber security assessment and education. Submitted, 2018.

[196] Robert Luh, Helge Janicke, and Sebastian Schrittwieser. AIDIS: Detecting and interpreting anomalous behavior in ubiquitous kernel processes. *Journal of Computers and Security (COSE)*, 2019.

[197] Xiapu Luo, Edmond WW Chan, and Rocky KC Chang. Vanguard: A new detection scheme for a class of TCP-targeted denial-of-service attacks. In *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP*, pages 507–518. IEEE, 2006.

[198] Neil MacDonald. Is Antivirus Obsolete?, 2015. URL `http://blogs.gartner.com/neil_macdonald/2012/09/13/is-antivirus-obsolete/`.

[199] Jennifer Mankin and David Kaeli. DIONE: a flexible disk monitoring and analysis framework. In *Research in Attacks, Intrusions, and Defenses*, pages 127–146. Springer, 2012.

[200] A. Marczewski. *Even Ninja Monkeys Like to Play: Gamification, Game Thinking and Motivational Design*. CreateSpace Independent Publishing Platform, 2015. ISBN 9781514745663.

[201] Stefan Marschalek, Robert Luh, Manfred Kaiser, and Sebastian Schrittwieser. Classifying malicious system behavior using event propagation trees. In *Proceedings of the 17th International Conference on Information Integration and Web-based Applications & Services*. Association for Computational Linguistics, 2015.

[202] Stefan Marschalek, Robert Luh, and Sebastian Schrittwieser. Endpoint data classification using markov chains. In *2017 International Conference on Software Security and Assurance (ICSSA)*, pages 56–59. IEEE, 2017.

[203] Abraham Harold Maslow. A theory of human motivation. *Psychological review*, 50(4):370, 1943.

[204] Sunu Mathew, Richard Giomundo, S. Upadhyaya, Moises Sudit, and Adam Stotz. Understanding multistage attacks by attack-track based visualization of heterogeneous event streams. In *Proceedings of the 3rd international workshop on Visualization for computer security*, pages 1–6. ACM, 2006.

[205] Sunu Mathew, Shambhu Upadhyaya, Moises Sudit, and Adam Stotz. Situation awareness of multistage cyber attacks by semantic event fusion. In *Military Communications Conference, 2010-MILCOM 2010*, pages 1286–1291. IEEE, 2010.

[206] Vasileios Mavroeidis and Siri Bromander. Cyber threat intelligence model: an evaluation of taxonomies, sharing standards, and ontologies within cyber threat intelligence. In *2017 European Intelligence and Security Informatics Conference (EISIC)*, pages 91–98. IEEE, 2017.

[207] David McClosky, Eugene Charniak, and Mark Johnson. Effective self-training for parsing. In *Proceedings of the main conference on human language technology conference of the North American Chapter of the Association of Computational Linguistics*, pages 152–159. Association for Computational Linguistics, 2006.

[208] Mary McCready and Mike Kinsman. Windowing functions (Azure Stream Analytics), 2019. URL https://docs.microsoft.com/en-us/stream-analytics-query/windowing-azure-stream-analytics.

[209] Deborah L. McGuinness, Frank Van Harmelen, and others. OWL web ontology language overview. *W3C recommendation*, 10(10):2004, 2004.

[210] Pritika Mehra. A brief study and comparison of snort and bro open source network intrusion detection systems. *International Journal of Advanced Research in Computer and Communication Engineering*, 1(6):383–386, 2012.

[211] Michael Meier. A model for the semantics of attack signatures in misuse detection systems. In *Information Security*, pages 158–169. Springer, 2004.

[212] Silvia Miksch and Wolfgang Aigner. A matter of time: Applying a data-users-tasks design triangle to visual analytics of time-oriented data. *Computers & Graphics*, 38:286–290, 2014. doi: 10.1016/j.cag.2013.11.002.

[213] Craig Miles, Arun Lakhotia, Charles LeDoux, Aaron Newsom, and Vivek Notani. VirusBattle: State-of-the-art malware analysis for better cyber threat intelligence. In *7th International Symposium on Resilient Control Systems*, pages 1–6. IEEE, 2014.

[214] Elinor Mills. A who's who of Mideast-targeted malware, 2015. URL http://www.cnet.com/news/a-whos-who-of-mideast-targeted-malware/.

[215] Li Ming and Paul Vitányi. *An introduction to Kolmogorov complexity and its applications*. Springer Heidelberg, 1997.

[216] Jelena Mirkovic and Peter Reiher. A taxonomy of DDoS attack and DDoS defense mechanisms. *ACM SIGCOMM Computer Communication Review*, 34(2):39–53, 2004.

[217] Natasha Arjumand Shoaib Mirza, Haider Abbas, Faheem Khan, and Jalal Al Muhtadi. Anticipating Advanced Persistent Threat (APT) countermeasures using collaborative security mechanisms. In *International Symposium on Biometrics and Security Technologies*, pages 129–132. IEEE, 2014.

[218] MIT Lincoln Laboratory. DARPA Intrusion Detection Evaluation, 2015. URL http://www.ll.mit.edu/ideval/data/.

[219] MITRE Corporation. CAPEC - Common Attack Pattern Enumeration and Classification (CAPEC), 2015. URL https://capec.mitre.org/.

[220] MITRE Corporation. Common Event Expression: CEE, A Standard Log Language for Event Interoperability in Electronic Systems, 2015. URL https://cee.mitre.org/.

[221] MITRE Corporation. CVE - Common Vulnerabilities and Exposures (CVE), 2015. URL https://cve.mitre.org/.

[222] MITRE Corporation. CWE - Common Weakness Enumeration, 2015. URL https://cwe.mitre.org/.

[223] MITRE Corporation. STIX - Structured Threat Information Expression | STIX Project Documentation, 2015. URL https://stixproject.github.io/.

[224] R Ann Miura-Ko, Benjamin Yolken, Nicholas Bambos, and John Mitchell. Security investment games of interdependent organizations. In *Communication, Control, and Computing, 2008 46th Annual Allerton Conference on*, pages 252–260. IEEE, 2008.

[225] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[226] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[227] Bamshad Mobasher, Robin Burke, and Jeff J Sandvig. Model-based collaborative filtering as a defense against profile injection attacks. In *AAAI*, volume 6, page 1388, 2006.

[228] Alexander M. Mood, F. A. Graybill, and Duane C. Boes. *Introduction to the Theory of Statistics*. McGraw-Hill, New York, 1974.

[229] Steve Morgan. 2017 Cybercrime Report. Technical report, Cybersecurity Ventures, 2017.

[230] Thomas J. Mowbray. *Cybersecurity: Managing Systems, Conducting Testing, and Investigating Intrusions*. John Wiley & Sons, October 2013. ISBN 978-1-118-84965-1.

[231] Christopher Munsey. Economic Espionage: Competing For Trade By Stealing Industrial Secrets, 2015. URL https://leb.fbi.gov/2013/october-november/economic-espionage-competing-for-trade-by-stealing-industrial-secrets.

[232] Gerhard Münz and Georg Carle. Real-time analysis of flow data for network attack detection. In *Integrated Network Management, 2007. IM'07. 10th IFIP/IEEE International Symposium on*, pages 100–108. IEEE, 2007.

[233] Katsuhiko Nakamura and Takashi Ishiwata. Synthesizing context free grammars from sample strings based on inductive CYK algorithm. In *International Colloquium on Grammatical Inference*, pages 186–195. Springer, 2000.

[234] Craig G. Nevill-Manning and Ian H. Witten. Identifying hierarchical strcture in sequences: A linear-time algorithm. *J. Artif. Intell. Res.(JAIR)*, 7:67–82, 1997.

[235] Kien C Nguyen, Tansu Alpcan, and Tamer Basar. Security games with incomplete information. In *Communications, 2009. ICC'09. IEEE International Conference on*, pages 1–6. IEEE, 2009.

[236] Caleb C Noble and Diane J Cook. Graph-based anomaly detection. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 631–636. ACM, 2003.

[237] Philip O'Kane, Sakir Sezer, and Kieran McLaughlin. Obfuscation: The hidden malware. *IEEE Security & Privacy*, 9(5):41–47, 2011.

[238] Xinming Ou, Raj Rajagopalan, and Sakthiyuvaraja Sakthivelmurugan. A practical approach to modeling uncertainty in intrusion analysis. Technical report, Technical report, Department of Computing and Information Sciences, Kansas State University, 2008.

[239] Jakub Pachocki, Greg Brockman, Jonathan Raiman, Susan Zhang, Henrique Pondé, Jie Tang, Filip Wolski, Christy Dennison, Rafal Jozefowicz, Przemyslaw Debiak, Brooke Chan, Szymon Sidor, David Farhi, and David Petrov. OpenAI Five, 2018. URL https://blog.openai.com/openai-five/.

[240] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up? Sentiment classification using machine learning techniques. In *Proceedings of the conference on Empirical methods in natural language processing*, pages 79–86. Association for Computational Linguistics, 2002. URL http://dl.acm.org/citation.cfm?id=1118704.

[241] Wonhyung Park and Seongjin Ahn. Performance comparison and detection analysis in snort and suricata environment. *Wireless Personal Communications*, 94 (2):241–252, 2017.

[242] Vern Paxson and Robin Sommer. The Zeek Network Security Monitor, 2019. URL https://www.zeek.org/.

[243] Bryan D. Payne, Martim Carbone, Monirul Sharif, and Wenke Lee. Lares: An architecture for secure active monitoring using virtualization. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pages 233–247. IEEE, 2008.

[244] Diego Perez-Botero, Jakub Szefer, and Ruby B. Lee. Characterizing hypervisor vulnerabilities in cloud computing servers. In *Proceedings of the International Workshop on Security in Cloud Computing*, pages 3–10. ACM, 2013.

[245] Leonid Peshkin. Structure induction by lossless graph compression. *arXiv preprint cs/0703132*, 2007. URL http://arxiv.org/abs/cs/0703132.

[246] Georgios Petasis, Georgios Paliouras, Vangelis Karkaletsis, Constantine Halatsis, and Constantine D Spyropoulos. e-grids: Computationally efficient gramatical inference from positive examples. *Grammars*, 7:69–110, 2004.

[247] Cynthia Phillips and Laura Painton Swiler. A graph-based system for network-vulnerability analysis. In *Proceedings of the 1998 workshop on New security paradigms*, pages 71–79. ACM, 1998.

[248] Ludovic Piètre-Cambacédès and Marc Bouissou. Attack and defense modeling with bdmp. In Igor Kotenko and Victor Skormin, editors, *Computer Network Security*, pages 86–101, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-14706-7.

[249] David Martin Powers. Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation. 2011.

[250] Rudy Prabowo and Mike Thelwall. Sentiment analysis: A combined approach. *Journal of Informetrics*, 3(2):143–157, 2009. URL http://www.sciencedirect.com/science/article/pii/S1751157709000108.

[251] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. URL http://www.R-project.org.

[252] Anand Rajaraman and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2011.

[253] Victor Raskin, Christian F. Hempelmann, Katrina E. Triezenberg, and Sergei Nirenburg. Ontology in information security: a useful theoretical foundation and methodological tool. In *Proceedings of the 2001 workshop on New security paradigms*, pages 53–59. ACM, 2001.

[254] Dan Raywood. Failure to detect Flame marks the end of signaturebased antivirus, 2015. URL http://www.scmagazineuk.com/failure-to-detect-flame-marks-the-end-of-signature-based-anti-virus/article/243505/.

[255] Abdul Razzaq, Hafiz Farooq Ahmed, Ali Hur, and Nasir Haider. Ontology based application level intrusion detection system by using bayesian filter. In *Computer, Control and Communication, 2009. IC4 2009. 2nd International Conference on*, pages 1–6. IEEE, 2009.

[256] Abdul Razzaq, Zahid Anwar, H. Farooq Ahmad, Khalid Latif, and Faisal Munir. Ontology for attack detection: An intelligent approach to web application security. *Computers & Security*, 45:124–146, September 2014. ISSN 01674048. doi: 10.1016/j.cose.2014.05.005.

[257] Abdul Razzaq, Khalid Latif, H. Farooq Ahmad, Ali Hur, Zahid Anwar, and Peter Charles Bloodsworth. Semantic security against web application attacks. *Information Sciences*, 254:19–38, January 2014. ISSN 00200255. doi: 10.1016/j.ins.2013.08.007.

[258] Jesse Read, Albert Bifet, Geoff Holmes, and Bernhard Pfahringer. Scalable and efficient multi-label classification for evolving data streams. *Machine Learning*, 88(1-2):243–272, 2012.

[259] Respect-IT. Kaos tutorial, 2015. URL http://www.objectiver.com/fileadmin/download/documents/KaosTutorial.pdf.

[260] Andreas Rieb and Ulrike Lechner. Operation digital chameleon: Towards an open cybersecurity method. In *Proceedings of the 12th International Symposium on Open Collaboration*, page 7. ACM, 2016.

[261] Andreas Josef Rieb, Marko Hofmann, Alexander Laux, Steffi Rudel, and Ulrike Lechner. Wie IT-Security Matchplays als Awarenessmaßnahme die IT-Sicherheit

verbessern können. In *13. Internationale Tagung Wirtschaftsinformatik*, pages 867–881, 2017.

[262] Konrad Rieck. Malheur - Automatic Analysis of Malware Behavior, 2015. URL http://www.mlsec.org/malheur/.

[263] Konrad Rieck, Philipp Trinius, Carsten Willems, and Thorsten Holz. Automatic analysis of malware behavior using machine learning. *J. Comput. Secur.*, 19(4): 639–668, December 2011. ISSN 0926-227X.

[264] Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

[265] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, pages 199–212. ACM, 2009.

[266] B Myerson Roger. Game theory: Analysis of conflict, 1991.

[267] Arpan Roy, Dong Seong Kim, and Kishor S Trivedi. Attack countermeasure trees (act): towards unifying the constructs of attack and defense trees. *Security and Communication Networks*, 5(8):929–943, 2012.

[268] Sankardas Roy, Charles Ellis, Sajjan Shiva, Dipankar Dasgupta, Vivek Shandilya, and Qishi Wu. A survey of game theory as applied to network security. In *System Sciences (HICSS), 2010 43rd Hawaii International Conference on*, pages 1–10. IEEE, 2010.

[269] Grzegorz Rozenberg. *Handbook of graph grammars and computing by graph transformation*, volume 1. World Scientific, 1997.

[270] Mark Russinovich. Process Monitor, 2015. URL https://technet.microsoft.com/en-us/sysinternals/bb896645.

[271] Mark E Russinovich, David A Solomon, and Alex Ionescu. *Windows internals*. Pearson Education, 2012.

[272] Alireza Sadighian, Saman Taghavi Zargar, José M. Fernandez, and Antoine Lemay. Semantic-based context-aware alert fusion for distributed Intrusion Detection Systems. In *Risks and Security of Internet and Systems (CRiSIS), 2013 International Conference on*, pages 1–6. IEEE, 2013.

[273] Yasubumi Sakakibara and Mitsuhiro Kondo. GA-based learning of context-free grammars using tabular representations. In *ICML*, volume 99, pages 354–360, 1999.

[274] Clemens Sauerwein, Christian Sillaber, Andrea Mussmann, and Ruth Breu. Threat intelligence sharing platforms: An exploratory study of software vendors and research perspectives. *Proceedings der 13. Internationalen Tagung Wirtschaftsinformatik*, 2017.

[275] Walter Scheirer and Mooi Choo Chuah. Syntax vs. semantics: competing approaches to dynamic network intrusion detection. *International Journal of Security and Networks*, 3(1):24–35, 2008.

[276] Bruce Schneier. Attack trees. *Dr. Dobb's journal*, 24(12):21–29, 1999.

[277] Bernhard Schölkopf, Robert C Williamson, Alex J Smola, John Shawe-Taylor, and John C Platt. Support vector method for novelty detection. In *Advances in neural information processing systems*, pages 582–588, 2000.

[278] Eresheim Sebastian, Luh Robert, and Schrittwieser Sebastian. The evolution of process hiding techniques in malware-current threats and possible countermeasures. *Journal of Information Processing*, 58(9), 2017.

[279] Seculert. Mahdi - The Cyberwar Savior?, 2015. URL http://www.seculert.com/blog/2012/07/mahdi-cyberwar-savior.html.

[280] M. Sedlmair, M. Meyer, and T. Munzner. Design study methodology: Reflections from the trenches and the stacks. 18(12):2431–2440, 2012. ISSN 1077-2626. doi: 10.1109/TVCG.2012.213.

[281] Yoav Seginer. Fast unsupervised incremental parsing. In *Annual Meeting-association for Computational Linguistics*, volume 45, page 384, 2007.

[282] Pavel Senin, Jessica Lin, Xing Wang, Tim Oates, Sunil Gandhi, Arnold P Boedihardjo, Crystal Chen, Susan Frankenstein, and Manfred Lerner. Grammarviz 2.0: a tool for grammar-based pattern discovery in time series. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 468–472. Springer, 2014.

[283] Pavel Senin, Jessica Lin, Xing Wang, Tim Oates, Sunil Gandhi, Arnold P Boedihardjo, Crystal Chen, and Susan Frankenstein. Time series anomaly discovery with grammar-based compression. In *EDBT*, pages 481–492, 2015.

[284] Y Shang. Optimal attack strategies in a dynamic botnet defense model. *Appl. Math. Inf. Sci*, 6:29–33, 2012.

[285] Claude Elwood Shannon. A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423, 1948.

[286] Paul Shannon, Andrew Markiel, Owen Ozier, Nitin S Baliga, Jonathan T Wang, Daniel Ramage, Nada Amin, Benno Schwikowski, and Trey Ideker. Cytoscape: A software environment for integrated models of biomolecular interaction networks. *Genome research*, 13(11):2498–2504, 2003.

[287] Monirul Sharif, Vinod Yegneswaran, Hassen Saidi, Phillip Porras, and Wenke Lee. Eureka: A framework for enabling static malware analysis. In *Computer security-ESORICS 2008*, pages 481–500. Springer, 2008.

[288] Ruey Shiang Shaw, Charlie C Chen, Albert L Harris, and Hui-Jou Huang. The impact of information richness on information security awareness training effectiveness. *Computers & Education*, 52(1):92–100, 2009.

[289] Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeannette M Wing. Automated generation and analysis of attack graphs. In *Security and privacy, 2002. Proceedings. 2002 IEEE Symposium on*, pages 273–284. IEEE, 2002.

[290] Adam Shostack. Elevation of privilege: Drawing developers into threat modeling. In *3GSE*, 2014.

[291] Michael Sikorski and Andrew Honig. *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. No Starch Press, 2012.

[292] Zach Solan, David Horn, Eytan Ruppin, and Shimon Edelman. Unsupervised learning of natural languages. *Proceedings of the National Academy of Sciences of the United States of America*, 102(33):11629–11634, 2005.

[293] Anil Somayaji and Stephanie Forrest. Automated response using system-call delay. In *Usenix Security Symposium*, pages 185–197, 2000.

[294] Aditya K. Sood and Richard J. Enbody. Targeted cyberattacks: a superset of advanced persistent threats. *IEEE security & privacy*, (1):54–61, 2013.

[295] A Sperotto, G Schaffrath, R Sadre, C Morariu, A Pras, and B Stiller. An Overview of IP Flow-Based Intrusion Detection. *IEEE Communications Surveys & Tutorials*, 12(3):343–356, 2010. ISSN 1553-877X. doi: 10.1109/SURV.2010.032210.00054.

[296] Stanford Center for Biomedical Informatics Research. Protégé, 2015. URL `http://protege.stanford.edu/`.

[297] Mark Steedman, Miles Osborne, Anoop Sarkar, Stephen Clark, Rebecca Hwa, Julia Hockenmaier, Paul Ruhlen, Steven Baker, and Jeremiah Crim. Bootstrapping statistical parsers from small datasets. In *Proceedings of the tenth conference on European chapter of the Association for Computational Linguistics-Volume 1*, pages 331–338. Association for Computational Linguistics, 2003.

[298] Andrew Stevenson and James R Cordy. A survey of grammatical inference in software engineering. *Science of Computer Programming*, 96:444–459, 2014.

[299] Gary Stoneburner, Alice Y. Goguen, and Alexis Feringa. SP 800-30. Risk Management Guide for Information Technology Systems. Technical report, NIST, 2002.

[300] Adam Stotz and Moises Sudit. Information fusion engine for real-time decision-making (INFERD): A perceptual system for cyber attack tracking. In *Information Fusion, 2007 10th International Conference on*, pages 1–8. IEEE, 2007.

[301] Chris Strasburg, Samik Basu, and Johnny S. Wong. S-MAIDS: A Semantic Model for Automated Tuning, Correlation, and Response Selection in Intrusion Detection Systems. pages 319–328. IEEE, July 2013. ISBN 978-0-7695-4986-6. doi: 10.1109/COMPSAC.2013.57.

[302] Richard S Sutton and Andrew G Barto. A temporal-difference model of classical conditioning. In *Proceedings of the ninth annual conference of the cognitive science society*, pages 355–378. Seattle, WA, 1987.

[303] Zareen Syed, Ankur Padia, Tim Finin, M Lisa Mathews, and Anupam Joshi. UCO: A unified cybersecurity ontology. 2016.

[304] Symantec. Regin: Top-tier espionage tool enables stealthy surveillance, 2015. URL http://www.symantec.com/connect/blogs/regin-top-tier-espionage-tool-enables-stealthy-surveillance.

[305] Symantec Security Response Team. Internet securty threat report, 2018.

[306] Urjita Thakar, Nirmal Dagdee, and Sudarshan Varma. Pattern Analysis and Signature Extraction For Intrusion Attacks On Web Services. *International Journal of Network Security & Its Applications*, 2(3):190–205, July 2010. ISSN 09752307. doi: 10.5121/ijnsa.2010.2313.

[307] The Hacker News. Harkonnen Operation — Malware Campaign that Went Undetected for 12 Years, 2015. URL http://thehackernews.com/2014/09/harkonnen-operation-malware-campaign_16.html.

[308] James J. Thomas and Kristin A. Cook, editors. *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. IEEE, 2005. ISBN 0769523234.

[309] Gerald R Thompson and Lori A Flynn. Polymorphic malware detection and identification via context-free grammar homomorphism. *Bell Labs Technical Journal*, 12(3):139–147, 2007.

[310] Mark E Thomson and Rossouw von Solms. Information security awareness: educating your users effectively. *Information management & computer security*, 6(4): 167–173, 1998.

[311] Kittikhun Thongkanchorn, Sudsanguan Ngamsuriyaroj, and Vasaka Visoottiviseth. Evaluation studies of three intrusion detection systems under various attacks and rule sets. In *2013 IEEE International Conference of IEEE Region 10 (TENCON 2013)*, pages 1–4. IEEE, 2013.

[312] Eric Totel, Bernard Vivinis, and Ludovic Mé. A language driven intrusion detection system for event and alert correlation. In *Proceedings at the 19th IFIP International Information Security Conference, Kluwer Academic, Toulouse*, pages 209–224. Springer, 2004.

[313] Brian Trammell and Benoit Claise. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information, 2015. URL https://tools.ietf.org/html/rfc7011.

[314] Philipp Trinius, Carsten Willems, Thorsten Holz, and Konrad Rieck. A malware instruction set for behavior-based analysis. Technical report, University of Mannheim, 2009.

[315] Alexey Tsymbal. The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin*, 106(2), 2004.

[316] UCLA Institute for Digital Research and Education. How are the likelihood ratio, Wald, and Lagrange multiplier (score) tests different and/or similar? URL https://stats.idre.ucla.edu/other/mult-pkg/faq/general/faqhow-are-the-likelihood-ratio-wald-and-lagrange-multiplier-score-tests-different-andor-similar/.

[317] Jeffrey Undercoffer, John Pinkston, Anupam Joshi, and Timothy Finin. A target-centric ontology for intrusion detection. In *18th International Joint Conference on Artificial Intelligence*, pages 9–15, 2004.

[318] University of California. KDD Cup 1999 Data, 2015. URL http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[319] University of Toronto. GRL Syntax, 2015. URL http://www.cs.toronto.edu/km/GRL/grl_syntax.html.

[320] Vijay K Vaishnavi and William Kuechler. *Design science research methods and patterns: innovating information and communication technology*. CRC Press, 2015.

[321] Menno Van Zaanen. ABL: Alignment-based learning. In *Proceedings of the 18th conference on Computational linguistics-Volume 2*, pages 961–967. Association for Computational Linguistics, 2000.

[322] Andrew Vance. Flow based analysis of Advanced Persistent Threats detecting targeted attacks in cloud computing. In *Infocommunications Science and Technology, 2014 First International Scientific-Practical Conference Problems of*, pages 173–176. IEEE, 2014.

[323] Lucretia H. Vanderwende and Donald Loritz. *The analysis of noun sequences using semantic information extracted from on-line dictionaries*. PhD thesis, Georgetown University, 1995.

[324] Celine Vens, Jan Struyf, Leander Schietgat, Sašo Džeroski, and Hendrik Blockeel. Decision trees for hierarchical multi-label classification. *Machine learning*, 73(2): 185, 2008.

[325] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al. StarCraft II: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017.

[326] W3C. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), 2015. URL http://www.w3.org/TR/soap12/.

[327] W3C. SPARQL 1.1 Overview, 2015. URL http://www.w3.org/TR/sparql11-overview/.

[328] W3C. Semantic web, 2015. URL http://www.w3.org/standards/semanticweb/.

[329] David Wagner and Paolo Soto. Mimicry attacks on host-based intrusion detection systems. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 255–264. ACM, 2002.

[330] Markus Wagner, Fabian Fischer, Robert Luh, Andrea Haberson, Alexander Rind, Daniel Keim, Wolfgang Aigner, Rita Borgo, Fabio Ganovelli, and Ivan Viola. A Survey of Visualization Systems for Malware Analysis. In *Eurographics Conference on Visualization (EuroVis) State of The Art Reports*, pages 105–125. EuroGraphics.

[331] Markus Wagner, Wolfgang Aigner, Alexander Rind, Hermann Dornhackl, Konstantin Kadletz, Robert Luh, and Paul Tavolato. Problem characterization and abstraction for visual analytics in behavior-based malware pattern analysis. In Kirsten Whitley, Sophie Engle, Lane Harrison, Fabian Fischer, and Nicolas Prigent, editors, *Proceedings 11th Workshop on Visualization for Cyber Security, VizSec*, pages 9–16. ACM, 2014. doi: 10.1145/2671491.2671498.

[332] Markus Wagner, Alexander Rind, Niklas Thür, and Wolfgang Aigner. A knowledge-assisted visual malware analysis system: Design, validation, and reflection of kamas. *Computers & Security*, 67:1–15, 02/2017 2017. ISSN 0167-4048. doi: 10.1016/j.cose.2017.02.003.

[333] Ju An Wang and Minzhe Guo. Ovm: an ontology for vulnerability management. In *Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies*, page 34. ACM, 2009.

[334] Rui Wang, Xiaoqi Jia, and Chujiang Nie. A Behavior Feature Generation Method for Obfuscated Malware Detection. pages 470–474. IEEE, August 2012. ISBN 978-0-7695-4719-0 978-1-4673-0721-5. doi: 10.1109/CSSS.2012.124.

[335] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8 (3-4):279–292, 1992.

[336] M. Wattenberg and F.B. Viegas. The word tree, an interactive visual concordance. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1221–1228, 2008. ISSN 1077-2626. doi: 10.1109/TVCG.2008.172.

[337] Martin Wattenberg. Arc diagrams: Visualizing structure in strings. In *Proceedings IEEE Symp. Information Visualization (InfoVis)*, pages 110–116, 2002. doi: {10.1109/INFVIS.2002.1173155}.

[338] Peter Wegner. Why interaction is more powerful than algorithms. *Communications of the ACM*, 40(5):80–91, 1997. ISSN 0001-0782. doi: 10.1145/253769.253801.

[339] Zikai Alex Wen, Yiming Li, Reid Wade, Jeffrey Huang, and Amy Wang. What.hack: Learn phishing email defence the fun way. In *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, CHI EA '17, pages 234–237, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4656-6. doi: 10.1145/3027063.3048412.

[340] Joshua S White, Thomas Fitzsimmons, and Jeanna N Matthews. Quantitative analysis of intrusion detection systems: Snort and suricata. In *Cyber Sensing 2013*, volume 8757, page 875704. International Society for Optics and Photonics, 2013.

[341] Janyce Wiebe, Theresa Wilson, Rebecca Bruce, Matthew Bell, and Melanie Martin. Learning subjective language. *Computational Linguistics*, 30(3):277–308, 2004.

[342] Carsten Willems, Thorsten Holz, and Felix Freiling. Toward automated dynamic malware analysis using cwsandbox. *IEEE Security & Privacy*, (2):32–39, 2007.

[343] Carsten Willems, Thorsten Holz, and Felix Freiling. Toward automated dynamic malware analysis using cwsandbox. *IEEE Security & Privacy*, (2):32–39, 2007.

[344] Glynn Winskel. *The formal semantics of programming languages: an introduction*. Foundations of computing. MIT Press, Cambridge, Mass., 5. print edition, 2001. ISBN 978-0-262-73103-4.

[345] Krist Wongsuphasawat and David Gotz. Outflow: Visualizing patient flow by symptoms and outcome. In *IEEE VisWeek Workshop on Visual Analytics in Healthcare, Providence, Rhode Island, USA*, 2011.

[346] Tobias Wüchner, Alexander Pretschner, and Martín Ochoa. DAVAST: data-centric system level activity visualization. pages 25–32. ACM Press, 2014. ISBN 978-1-4503-2826-5. doi: 10.1145/2671491.2671499.

[347] Haizhi Xu, Wenliang Du, and Steve J. Chapin. Context sensitive anomaly monitoring of process control flow to detect mimicry attacks and impossible paths. In *RAID*, pages 21–38. Springer, 2004.

[348] Wei Yan, Edwin Hou, and Nirwan Ansari. Extracting attack knowledge using principal-subordinate consequence tagging case grammar and alerts semantic networks. In *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*, pages 110–117. IEEE, 2004.

[349] Wei Yan, Edwin Hou, and Nirwan Ansari. A description logic based approach for IDS security information management. In *Advances in Wired and Wireless Communication, 2005 IEEE/Sarnoff Symposium on*, pages 25–28. IEEE, 2005.

[350] Xifeng Yan and Jiawei Han. gspan: Graph-based substructure pattern mining. In *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*, pages 721–724. IEEE, 2002.

[351] Xia Zheng You and Zhang Shiyong. A kind of network security behavior model based on game theory. In *Parallel and Distributed Computing, Applications and Technologies, 2003. PDCAT'2003. Proceedings of the Fourth International Conference on*, pages 950–954. IEEE, 2003.

[352] Apostolis Zarras, Antonis Papadogiannakis, Robert Gawlik, and Thorsten Holz. Automated generation of models for fast and precise detection of HTTP-based malware. In *Privacy, Security and Trust (PST), 2014 Twelfth Annual International Conference on*, pages 249–256. IEEE, 2014.

[353] Zhiping Zeng, Anthony KH Tung, Jianyong Wang, Jianhua Feng, and Lizhu Zhou. Comparing stars: On approximating graph edit distance. *Proceedings of the VLDB Endowment*, 2(1):25–36, 2009.

[354] Liangwei Zhang, Jing Lin, and Ramin Karim. Adaptive kernel density-based anomaly detection for nonlinear systems. *Knowledge-Based Systems*, 139:50–63, 2018.

[355] Min-Ling Zhang and Zhi-Hua Zhou. ML-KNN: A lazy learning approach to multi-label learning. *Pattern recognition*, 40(7):2038–2048, 2007.

[356] Qinghua Zhang, Douglas S. Reeves, Peng Ning, and S. Purushothaman Iyer. Analyzing network traffic to detect self-decrypting exploit code. In *Proceedings of the 2nd ACM symposium on Information, computer and communications security*, pages 4–12. ACM, 2007.

[357] Chunying Zhao, Jun Kong, and Kang Zhang. Program behavior discovery and verification: A graph grammar approach. *IEEE Transactions on software Engineering*, 36(3):431–448, 2010.

[358] Bin Zhu and Ali A. Ghorbani. *Alert correlation for extracting attack strategies*. PhD thesis, Citeseer, 2005.

[359] Detlef Zimmer and Rainer Unland. On the semantics of complex events in active database management systems. In *Data Engineering, 1999. Proceedings., 15th International Conference on*, pages 392–399. IEEE, 1999.

Acronyms

General glossary of acronyms, alphabetically sorted:

API	Application Programming Interface
APT	Advanced Persistent Threat
ATA	Advanced Targeted Attack (see also APT)
AV	Anti-Virus
C2	Command and Control
CAPEC	Common Attack Pattern Enumeration and Classification
CIA	Confidentiality, Integrity, Availability
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerabilities Scoring System
CWE	Common Weakness Enumeration
DHCP	Dynamic Host Configuration Protocol
DLL	Dynamic Link Library
DNS	Domain Name System
DMZ	De-Militarized Zone
DoS	Denial of Service
ICS	Industrial Control System
ICT	Information and Communications Technology
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
LAN	Local Area Network
LLR	Log Likelihood Ratio
LR	Likelihood Ratio
NDA	Non-Disclosure Agreement
NIST	National Institute of Standards and Technology
NVD	National Vulnerability Database
OS	Operating System
PID	Process Identifier (Windows)
PLC	Programmable Logic Controller
RF	Random Forest
RL	Reinforcement Learning
RPG	Role-Playing Game
SCADA	Supervisory Control and Data Acquisition

SIEM	Security Information and Event Management
SQL	Structured Query Language
SSDT	System Service Descriptor Table (Windows)
STIX	Structured Threat Information Expression
SVM	Support Vector Machine
TA	Targeted Attack (see also ATA)
TAON	Targeted Attack Ontology [188]
TID	Thread Identifier (Windows)
VM	Virtual Machine
VPN	Virtual Private Network
VoIP	Voice over IP

Acronyms in the context of the PenQuest game model introduced in Chapter 7, sorted by type and occurrence:

SO	Sophistication (Actor skill level attribute)
DE	Determination (Actor motivation attribute)
WE	Wealth (Actor resources attribute)
INI	Initiative (Actor time efficiency derived attribute)
INS	Insight (Opponent knowledge derived attribute)
TH	Thief (STIX Cyber Espionage Operations actor)
EX	Explorer (STIX Hacker, White Hat actor)
RO	Rogue (STIX Hacker, Grey Hat actor)
RA	Raider (STIX Hacker, Black Hat actor)
CR	Crusader (STIX Hacktivist actor)
OP	Operative (STIX State Actor/Agency actor)
IN	Infiltrator (STIX Insider Threat actor)
PR	Protester (STIX Disgruntled Customer actor)
CP	Production Company (Primary sector actor)
CM	Manufacturing Company (Secondary sector actor)
CS	Services Company (Tertiary sector actor)
IF	Infrastructure provider (Actor)
MI	Military (Actor)
SA	State Actor/Agency (Actor)
ED	Education sector (Actor)
PI	Private Individual (Actor)
ATT	Attacker equipment (Enabler)
MPT	Multi-Purpose Tool (ATT)
Sca	System scanner (ATT)
VSc	Vulnerability Scanner (ATT)
NSc	Network Scanner (ATT)
Wir	Wireless tool (ATT)
Pwd	Password cracker (ATT)

Mal	Malware (ATT)
VUL	Vulnerability (Enabler)
PoC	Proof of Concept (Enabler Maturity)
AST	Asset (Disabler)
SEC	Security solution (Disabler)
Pre	Prevention solution (SEC)
Det	Detection solution (SEC)
Del	Delay solution (SEC)
Rec	Recovery solution (SEC)
Cnt	Countermeasures (SEC)
POL	Policy (Disabler)
FIX	Fix for VUL (Disabler)
***DC, D**	Detection Chance (Effect)
***SC, S**	Success Chance (Effect)
***CR**	Credits (Effect)
***STA**	Status (Effect)
H	Host (Effect target)
M	Mobile system (Effect target)
I	Industrial system (Effect target)
N	Network asset (Effect target)
T	Third party service (Effect target)
R.*	Reconnaissance (APT kill chain phase)
W.*	Weaponization (APT kill chain phase)
D.*	Delivery (APT kill chain phase)
E.*	Exploitation (APT kill chain phase)
I.*	Installation (APT kill chain phase)
C.*	Command and Control (APT kill chain phase)
A.*	Action on Objective (APT kill chain phase)
IG	Information Gathering (CAPEC primary attack class)
IN	Injection (CAPEC primary attack class)
SE	Social Engineering (CAPEC primary attack class)
SA	State Attack (CAPEC primary attack class)
FA	Function Abuse (CAPEC primary attack class)
BF	Brute Force (CAPEC primary attack class)
IA	Illegal Access (CAPEC primary attack class)
DM	Data Manipulation (CAPEC primary attack class)
PR	Preparation (Non-CAPEC primary attack class)
CO	Communication (Non-CAPEC primary attack class)
IL	Information Leakage protection (Primary control class)
CP	Context Protection (Primary control class)
AW	Awareness (Primary control class)
SP	State Protection (Primary control class)
FI	Function Integrity (Primary control class)

AP	Authentication Protection (Primary control class)
AC	Access Control (Primary control class)
DI	Data Integrity (Primary control class)
SI	Security Intelligence (Primary control class)
CS	Communications Security (Primary control class)
ACM	Account Management (NIST-based defense action)
ACE	Access Enforcement (NIST-based defense action)
IFE	Information Flow Enforcement (NIST-based defense action)
LEP	Least Privilege (NIST-based defense action)
REA	Remote Access (NIST-based defense action)
RST	Role-based Security Training (NIST-based defense action)
COM	Continuous Monitoring (NIST-based defense action)
CCC	Configuration Change Control (NIST-based defense action)
COS	Configuration Settings (NIST-based defense action)
COP	Contingency Plan (NIST-based defense action)
AUM	Authenticator Management (NIST-based defense action)
INH	Incident Handling (NIST-based defense action)
NOM	Nonlocal Maintenance (NIST-based defense action)
MES	Media Storage (NIST-based defense action)
SEP	Security Engineering Principles (NIST-based defense action)
BOP	Boundary Protection (NIST-based defense action)
CRP	Cryptographic Protection (NIST-based defense action)
FLR	Flaw Remediation (NIST-based defense action)
MCP	Malicious Code Protection (NIST-based defense action)
ISM	Information System Monitoring (NIST-based defense action)

Authorship Confirmation

As a co-author of one or several of the the below listed papers...

- [187] Robert Luh, Stefan Marschalek, Manfred Kaiser, Helge Janicke, and Sebastian Schrittwieser. Semantics-aware detection of targeted attacks: a survey. Journal of Computer Virology and Hacking Techniques, pages 1–39, 2016.
- [190] Robert Luh, Sebastian Schrittwieser, and Stefan Marschalek. LLR-based sentiment analysis for kernel event sequences. In Proceedings of the 31st International Conference on Advanced Information Networking and Applications (AINA), pages 764–771. IEEE, 2017.
- [193] Robert Luh, Gregor Schramm, Markus Wagner, Helge Janicke, and Sebastian Schrittwieser. SEQUIN: a grammar inference framework for analyzing malicious system behavior. Journal of Computer Virology and Hacking Techniques, pages 1–21, 2018.
- [195] Robert Luh, Marlies Temper, Simon Tjoa, Sebastian Schrittwieser, and Helge Janicke. PenQuest: A Gamified Attacker/Defender Meta Model for Cyber Security Assessment and Education. Journal of Computer Virology and Hacking Techniques, Springer, *submitted*, 2018.
- [196] Robert Luh, Helge Janicke, and Sebastian Schrittwieser. AIDIS: Detecting and Classifying Anomalous Behavior in Ubiquitous Kernel Processes. Journal of Computers and Security (COSE), Elsevier, 2019.

...I hereby confirm that, in accordance to DMU regulations, Robert Luh is the person responsible for producing the first draft, and that he is author of at least 75% of the final content of the respective paper.

Helge Janicke	_____	Stefan Marschalek	_____
Gregor Schramm	_____	Sebastian Schrittwieser	_____
Marlies Temper	_____	Simon Tjoa	_____
Markus Wagner	_____		